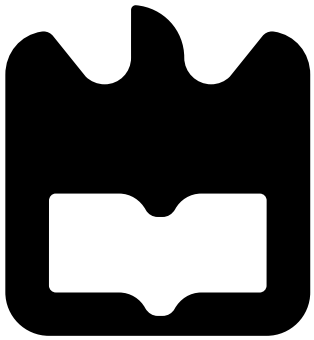
Universidade de Aveiro
Departamento de
Electrónica, Telecomunicações e Informática
2011

Carlos Eduardo
Braga Ameixieira

Desenvolvimento da Sub-Camada MAC IEEE 802.11p/1609.4



Carlos Eduardo Braga Ameixieira

Desenvolvimento da Sub-Camada MAC IEEE 802.11p/1609.4

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Doutor Arnaldo Silva Rodrigues de Oliveira e da Doutora Susana Isabel Barreto de Miranda Sargento, Professores Auxiliares do Departamento Electrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president

Prof. Doutor João Nuno Pimentel da Silva Matos
Professor Associado da Universidade de Aveiro (por delegação da Reitora da Universidade de Aveiro)

vogais / examiners committee

Prof. Doutor Luis Miguel Pinho de Almeida
Professor Associado do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto (Arguente)

Prof. Doutor Arnaldo Silva Rodrigues de Oliveira
Professora Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro (Orientador)

Prof. Doutora Susana Isabel Barreto de Miranda Sargento
Professora Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro (Co-orientadora)

agradecimentos / acknowledgements

Primeiramente agradeço à pessoa a que devo tudo o que sou e tudo o que conquistei até hoje, a minha mãe, Tereza de Jesus Braga Ameixieira, pois aturou todos os meus altos e baixos e sem ela não estaria onde estou. Agradeço também à minha irmã, Fernanda Braga Ameixieira, que todos estes anos me apoiou incondicionalmente. Ao meu tio José Mauro, à minha tia Sandra, aos meus primos André e Diogo, e ao meu padrinho Renato Lucas, um sincero obrigado.

Num patamar imediatamente a seguir encontram-se alguns amigos, poucos mas muito bons, com quem partilhei os meus momentos académicos, e não só: João Vieira (Peixe), João Francisco Graça (Fran), Bruno Carvalho, João Nunes (Chavetas), Ana Pires e Ana Machado. Obrigado pelo apoio.

No que diz respeito a este trabalho, devo muito ao meu orientador o Professor Doutor Arnaldo Oliveira, por tudo o que me ensinou e por todas as críticas sempre construtivas que fez. Ao Engenheiro José Pedro Matos, que me impulsionou e guiou no decorrer deste trabalho, com quem aprendi muito. Ao amigo Mestre Engenheiro Ricardo Moreira, por tornar o ambiente de trabalho tão confortável e pelo apoio todos os dias, obrigado.

Agradeço também a Professora Doutora Susana Sargento pela oportunidade que me ofereceu de trabalhar no projecto DRIVE-IN e ao Mestre André Cardote.

Palavras-chave

VANET, Comunicações Veiculares, norma IEEE 802.11p, norma IEEE 1609.4, Comunicação Veículo-a-Veículo, Coordenação de canais, Subcamada MAC, WAVE, DSRC.

Resumo

As comunicações veiculares visam melhorar a segurança nas estradas através da disponibilização de informação ao condutor e oferecer aos restantes ocupantes uma viagem mais agradável através de fontes de informação e entretenimento. Este cenário permite, por exemplo, o envio de mensagens de emergência, assim como o envio de um serviço disponibilizado pela internet. Esta forma de comunicação denominada *Vehicular Ad-hoc NETwork* (VANET), é um tipo de comunicação em que os dispositivos cooperam entre si, sem a necessidade da existência de um ponto de controlo fixo. A VANET apresenta as vantagens de possuir uma topologia de rede variável, permitir a comunicação entre um vasto número de nós e adaptação a percursos específicos.

A família de normas IEEE 1609.x e a norma IEEE 802.11p especificam cada camada do modelo *Open Systems Interconnection* (OSI) que implementam as VANET's. Este conjunto de normas constitui o núcleo da *Dedicated Short-Range Communications* (DSRC).

O trabalho descrito nesta dissertação insere-se no projecto *Distributed Routing and Infotainment through VEhicular Inter-Networking* (DRIVE-IN). Este projecto tem como objectivo avaliar como a comunicação *Vehicle to Vehicle* (V2V) pode melhorar a experiência do utilizador nas estradas e a eficiência global da utilização do veículo e das estradas, através da realização de uma *testbed* na cidade do Porto, utilizando 465 taxis e alguns autocarros locais para tal. Este trabalho insere-se na primeira fase do projecto DRIVE-IN, onde todo o sistema base para VANET é desenvolvido.

O trabalho realizado no âmbito desta dissertação teve como foco a implementação das normas IEEE 802.11p e IEEE 1609.4, que descrevem o comportamento da sub-camada *Medium Access Control* (MAC) para dispositivos *Wireless Access in the Vehicular Environment* (WAVE), sub-camada entre a camada física e a sub-camada *Logical Link Control* (LLC) do modelo OSI, responsável pelo controlo do acesso ao meio físico. A norma IEEE 802.11p apresenta alterações em relação à *Quality of Service* (QoS) para dispositivos WAVE e introduz o conceito da comunicação fora do contexto de uma *Basic Service Set* (BSS). A norma IEEE 1609.4 introduz os conceitos de *Service CHannel* (SCH) e *Control CHannel* (CCH), e descreve o funcionamento da sub-camada MAC WAVE. Esta norma descreve o encaminhamento, a coordenação, os tipos de acesso aos canais e a sincronização.

Na implementação das funcionalidades apresentadas nestas normas, foi utilizada uma placa wireless mini-*Peripheral Component Interconnect* (PCI) que utiliza o *chipset* da *Atheros* AR5414A. O driver dependente do hardware utilizado para esta placa é denominado *ath5k*. Através da alteração do *Free and Open Source Software* (FOSS) *driver ath5k*, foi implementada a sub-camada MAC WAVE em Linux. Este driver faz a interface entre o hardware e a *stack* wireless do Linux, estando esta dividida nos módulos *mac80211* e *cfg80211*. Utilizou-se a aplicação *iw* para facilitar a configuração da MAC, visto que *iw* utiliza *sockets netlink* para comunicação *userspace-kernelspace*.

Por último são apresentados os resultados conseguidos, tais como o tempo médio de troca entre canais e taxas de transferência reais conseguidas com esta implementação.

Keywords

VANET, Vehicular Communication, IEEE 802.11p standard, IEEE 1609.4 standard, Vehicle-to-Vehicle Communication, Channel Coordination, MAC Sub-layer, WAVE, DSRC.

Abstract

The vehicular communications aim to improve safety on the roads by providing information to the driver and offering to the rest of the occupants a more enjoyable trip through sources of information and entertainment. This scenario allows, for example, the sending of warning messages, as well as services available over the internet. This form of communication named *Vehicular Ad-hoc NETwork* (VANET), is a type of communication that devices cooperate with each other, without need to exist a fixed control point. VANET presents the advantages of owning a variable network topology, allowing communication between a large number of nodes and adaptation to specific routes.

IEEE 1609.x standards family and IEEE Std 802.11p specify each layer of the model that implement the *Open Systems Interconnection* (OSI) model for VANET's. This set of standards is the core of the *Dedicated Short-Range Communications* (DSRC).

The work described in this thesis is part of the *Distributed Routing and Infotainment through Vehicular Inter-Networking* (DRIVE-IN). This project aims to evaluate how communication *Vehicle to Vehicle* (V2V) can improve the user experience on the roads and the overall efficiency of the use of testbed in Porto, using 465 taxis and some local buses. This work is part of the first phase of the DRIVE-IN project, where the entire base system for VANET is developed.

The work done in this thesis focused on the implementation of IEEE Std 802.11p e IEEE Std 1609.4, which describe the behavior of the *Medium Access Control* (MAC) sub-layer to *Wireless Access in the Vehicular Environment* (WAVE) devices, sub-layer between the physical layer and the *Logical Link Control* (LLC) sub-layer of the OSI model, responsible for controlling access to the physical environment. IEEE Std 802.11p shows changes in *Quality of Service* (QoS) for WAVE devices and introduces the concept of communication outside the context of a *Basic Service Set* (BSS). IEEE Std 1609.4 introduces the concepts of *Service CHannel* (SCH) and *Control CHannel* (CCH), and describes the operation of the WAVE MAC sub-layer. This standard describes the channel routing, coordination and access modes, and synchronization. In the implementation of the functionalities described by these standards, the mini-*Peripheral Component Interconnect* (PCI) wireless board, that uses the AR5414A *Atheros* chipset, was used. The hardware-dependent driver used for this board is called *ath5k*. By changing the *Free and Open Source Software* (FOSS) *driver ath5k*, the MAC WAVE sub-layer was implemented in Linux.

The hardware interfaces with the wireless stack through this driver, while wireless stack is divided in *mac80211* and *cfg80211* modules. *iw* application was used to facilitate the setting of the MAC sub-layer, since *iw* uses *netlink sockets* to *userspace-kernelspace* communication. Finally the achieved results are presented, such as the average channel switching time and the actual transfer rates achieved with this implementation.

Conteúdo

Lista de Figuras

Lista de Tabelas

Acrónimos

AC *Access Category*

ACI *Access Category Index*

AIFS *Arbitration Inter-Frame Space*

AIFSN *Arbitration Inter-Frame Space Number*

AP *Access Point*

API *Application Programming Interface*

BSS *Basic Service Set*

CAN *Controller Area Network*

CCH *Control CHannel*

CLI *Command-Line Interface*

CPU *Central Processing Unit*

CW *Contention Window*

DC *Direct Current*

DCD *Data Carrier Detect*

DDR *Double Data Rate*

DRAM *Dynamic Random Access Memory*

DSRC *Dedicated Short-Range Communications*

EDCA *Enhanced Distributed Channel Access*

FOSS *Free and Open Source Software*

GPIO *General Purpose Input/Output*

GPS *Global Positioning System*

I2V *Infrastructure to Vehicle*

IEEE *Institute of Electrical and Electronics Engineers*

IP *Internet Protocol*

IPv6 *Internet Protocol version 6*

IPC *Inter-Process Comunication*

ITS *Intelligent Transportation Systems*

LLC *Logical Link Control*

LPC *Low Pin Count*

MAC *Medium Access Control*

MANET *Mobile Ad-hoc NETwork*

MIB *Management Information Base*

MLME *MAC subLayer Management Entity*

MLMEX *MAC subLayer Management Entity Extension*

MPDU *MAC Protocol Data Unit*

MSDU *MAC Service Data Unit*

NMEA *National Marine Electronics Association*

OBU *On Board Unit*

OSI *Open Systems Interconnection*

PC *Personal Computer*

PCI *Peripheral Component Interconnect*

PCI-E *Peripheral Component Interconnect Express*

PCMCIA *Personal Computer Memory Card International Association*

PHY *Physical Layer*

PID *Proportional Integral Derivative*

PLME *PHY subLayer Management Entity*

POE *Power Over Ethernet*

PPS *Pulse Per Second*

QoS *Quality of Service*

RCPI *Received Channel Power Indicator*

RF *Radio Frequency*

RS-232 *Recommended Standard-232*

RSU *Road Side Unit*

SAP *Service Access Point*

SCH *Service CHannel*

SIFS *Short Inter-Frame Space*

STA *Station*

TA *Timing Advertisement frame*

TSF *Timing Synchronization Function*

TXOP *Transmit Opportunity*

UDP *User Datagram Protocol*

UP *User Priority*

USB *Universal Serial Bus*

UTC *Universal Time Coordinated*

V2V *Vehicle to Vehicle*

V2I *Vehicle to Infrastructure*

V2X *Vehicle to anything*

VANET *Vehicular Ad-hoc NETwork*

VGA *Video Graphics Array*

VSA *Vendor Specific Action frame*

WAVE *Wireless Access in the Vehicular Environment*

WDS *Wireless Distribution System*

WEXT *Wireless-Extensions*

WLAN *Wireless Local Area Network*

WME *WAVE Management Entity*

WSA *WAVE Service Advertisement*

WSM *WAVE Short Message*

WSMP *WAVE Short Message Protocol*

WSU *Wireless Safety Unit*

Capítulo 1

Introdução

1.1 Enquadramento

A comunicação veicular consiste na comunicação entre dois veículos, *Vehicle to Vehicle* (V2V), ou na comunicação entre um veículo e uma infraestrutura colocada em algum ponto da estrada, *Vehicle to Infrastructure* (V2I), ou *Infrastructure to Vehicle* (I2V) no sentido inverso. De uma forma geral, podemos dizer que as comunicações veiculares consistem na comunicação entre o veículo e qualquer dispositivo preparado para tal, ao longo do seu trajecto, *Vehicle to anything* (V2X). A comunicação veicular torna-se possível através da troca de informação entre dispositivos colocados nos veículos, *On Board Unit* (OBU), e equipamentos estáticos em pontos específicos da estrada, *Road Side Unit* (RSU). Este dois tipos de dispositivos constituem os nodos da rede veicular e estão destinados a cooperar entre si de modo a garantir conectividade eficiente ao utilizador.

As comunicações veiculares constituem um novo tipo de comunicação destinado à prevenção de situações de perigo, congestionamento ou condições meteorológicas nas estradas, ao pagamento de taxas ao longo de um trajecto e ao entretenimento do utilizador, como, por exemplo, a transmissão de um vídeo ou mesmo acesso a serviços disponibilizados via internet.

A comunicação entre veículos apresenta a necessidade de haver comunicação móvel e sem a necessidade da existência obrigatória de qualquer infraestrutura que controle o encaminhamento de pacotes na rede. Esta forma de comunicação é denominada por *Vehicular Ad-hoc NETwork* (VANET).

VANET é uma classe particular das *Mobile Ad-hoc NETwork* (MANET) para ambientes veiculares. Como se pode imaginar através da análise das diferentes situações encontradas por um veículo ao longo do seu trajecto, torna-se praticamente impossível haver uma infraestru-

tura fixa sempre disponível para contribuir na gestão da rede. Sendo assim, neste tipo de comunicação há a necessidade dos diferentes dispositivos cooperarem durante o seu trajecto [1].

Uma VANET é caracterizada pelos seguintes aspectos [2]:

- Uma topologia de rede bastante variável, apesar de restrita, visto que apresenta comunicação apenas entre veículos ou entre veículos e dispositivos colocados na beira da estrada;
- Possibilita a comunicação entre um vasto número de nós. Note-se que quanto maior for o número de nós na rede, maior será a probabilidade de perda de pacotes recebidos por cada nó;
- Condições precárias de comunicação, uma vez que podem haver prédios e outros obstáculos a bloquear o sinal;
- Adaptação aos percursos padrão específicos dos veículos. A comunicação torna-se mais fiável em zonas de mais alta mobilidade;
- Não apresenta preocupações relevantes em relação à energia necessária à alimentação dos dispositivos.

De modo a proceder à implementação de VANETs foi necessário desenvolver e adaptar protocolos e normas. Como qualquer género de comunicação *wireless*, as VANETs devem obedecer a normas de modo a respeitar regras bem definidas, sendo possível a compreensão da sua implementação de forma globalizada.

Com o objectivo de se conseguir uma implementação eficiente, várias normas foram publicadas ao longo dos anos, na sua grande maioria pelo IEEE. O conjunto base de normas que visa a implementação das comunicações veiculares é designado por *Dedicated Short-Range Communications* (DSRC) 5.9 GHz. Actualmente, o DSRC 5.9 GHz é considerado o conjunto de normas *wireless* mais promissor, sendo composto pela norma IEEE 802.11p [3] e o conjunto de normas IEEE 1609.x, denominado *Wireless Access in the Vehicular Environment* (WAVE). A norma IEEE 802.11 [4] apresenta as especificações necessárias à implementação dos modos *wireless* mais comuns, como o modo *ad-hoc*.

De modo a suportar aplicações *Intelligent Transportation Systems* (ITS), foram especificadas alterações à norma IEEE 802.11. A troca de informação entre veículos a altas velocidades e entre veículos e dispositivos colocados na beira da estrada que operem na banda de

frequência 5.850-5.925 GHz, definida para DSRC 5.9 GHz, são especificadas na revisão da norma IEEE 802.11, que deu origem à norma IEEE 802.11p. A norma IEEE 802.11p apresenta, então, algumas alterações necessárias aos modos de operação descritos na norma IEEE 802.11, de modo a suportar a comunicação entre nodos num ambiente veicular. Contudo, as alterações definidas na norma IEEE 802.11p apenas oferecem um ponto de partida para uma implementação eficiente da comunicação entre veículos, que conduziu à publicação das normas que constituem o WAVE.

A norma IEEE 1609.4 [5] constitui a parte do WAVE que se foca na implementação da sub-camada *Medium Access Control* (MAC) [6]. Esta norma parte do ponto onde termina a norma IEEE 802.11p, e divide a banda de frequência 5.850-5.925 GHz, pelos vários canais de serviço (*Service CHannel* (SCH)) e pelo canal de controlo (*Control CHannel* (CCH)), como ilustrado pela figura 1.1. Perante os tipos de canais definidos, a norma IEEE 1609.4 especifica os modos de acesso aos canais e funcionalidades relativas a cada canal. As normas IEEE 802.11p e IEEE 1609.4, constituem, portanto, o conjunto principal de normas às quais a implementação do trabalho descrito por esta dissertação deve respeitar.

Número do canal	172	174	176	178	180	182	184
	SCH	SCH	SCH	CCH	SCH	SCH	SCH
Frequência (GHz)	5.86	5.87	5.88	5.89	5.9	5.91	5.92

Figura 1.1: Divisão da banda de frequência 5.9 GHz por tipo de canais, definido na norma IEEE 1609.4

A figura 1.2 ilustra o panorama obtido com a implementação do DSRC, com a interacção entre veículos equipados com dispositivos WAVE, OBU, e entre veículos e infraestruturas que contém dispositivos estáticos em pontos específicos da estrada, RSU.

A sub-camada MAC é inserida como sub-camada da camada de ligação de dados (*Data-Link layer*), segunda camada do modelo *Open Systems Interconnection* (OSI), ilustrado pela figura 1.3. Segundo este modelo, a sub-camada MAC controla o acesso ao meio físico, funcionando como intermediário de decisão, se múltiplos dispositivos competirem ao mesmo tempo ao acesso à mesma ligação física. A sub-camada MAC situa-se, portanto, entre a camada física, primeira camada do modelo OSI, e a sub-camada *Logical Link Control* (LLC) da camada de ligação de dados [7].

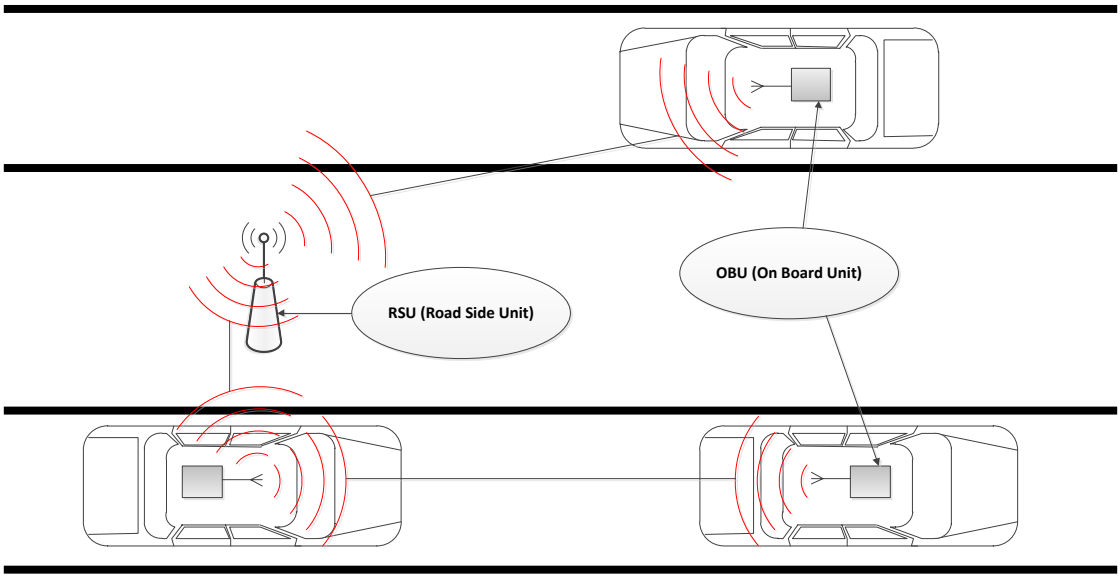


Figura 1.2: DSRC - *Dedicated Short-Range Communications*

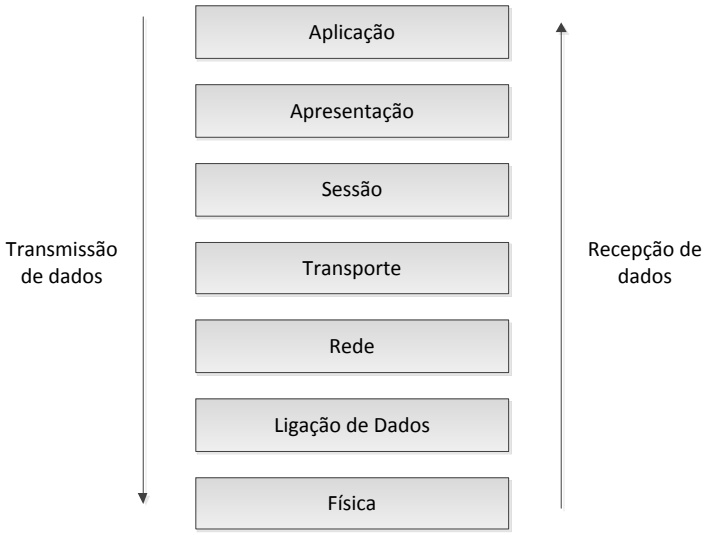


Figura 1.3: Sete camadas do modelo OSI

No cenário das comunicações veiculares, a sub-camada MAC desempenha a função indi-

cada, coordenando o acesso aos diferentes canais, sincronizando os mesmos, e garantindo comunicação entre a camada física e a sub-camada LLC, com níveis de prioridade bem definidos.

A implementação da sub-camada MAC WAVE, respeitando as normas IEEE 802.11p/ 1609.4, pode seguir as seguintes abordagens:

- Desenvolvimento do hardware e implementação de todo o conjunto de normas e protocolos necessários à implementação do projecto.
- Implementação de todo o conjunto de normas e protocolos necessários à implementação do projecto.
- Utilização de um *driver* desenvolvido para o meio *wireless*, procedendo-se à alteração do mesmo de forma a atingir as exigências descritas nas normas.

A escolha da abordagem a seguir deve ser feita consoante os objectivos do projecto, visto que o tempo que cada abordagem requer para a sua implementação varia de forma significativa. A primeira abordagem, por exemplo, implica o desenvolvimento de todo o sistema, o que não é fiável para um projecto onde o tempo de desenvolvimento e o custo são determinantes.

Esta dissertação é parte integrante do projecto de investigação *Distributed Routing and Infotainment through VEhicular Inter-Networking* (DRIVE-IN), financiado pelo programa Carnegie Mellon University Portugal, através da Fundação para a Ciência e Tecnologia. O objectivo principal do projecto DRIVE-IN é investigar como a comunicação V2V pode melhorar as condições que o utilizador dispõe nas estradas. Este projecto apresenta uma fase inicial em que o equipamento e respectivas aplicações são preparados e implementados para a realização de uma *testbed* na cidade do Porto, em que serão equipados 465 taxis.

1.2 Motivação

O grupo IEEE publicou uma versão da norma IEEE 1609.4 em 2010 que constitui a revisão da publicação lançada em 2006. Facilmente percebe-se, por ser uma versão tão recente, que neste momento não existe nenhum produto no mercado que respeite na íntegra esta revisão, que apresenta muitas diferenças em relação à antiga publicação. Assim, esta apresenta-se como uma grande motivação deste projecto, visto que será um dos primeiros a apresentar uma implementação da revisão da norma IEEE 1609.4.

Além da escassa implementação, não são muitos os equipamentos que dão o suporte necessário à implementação da sub-camada MAC WAVE, visto que exige a comunicação

na banda de frequência dos 5.9 GHz. Os equipamentos que existem apresentam um custo elevado, normalmente entre 900 a 1900 euros por unidade, dependendo da quantidade requerida, e um dos pontos fortes do projecto DRIVE-IN visa um baixo custo através da utilização de uma placa mini-*Peripheral Component Interconnect* (PCI) e de um micro-*Personal Computer* (PC) que suporta a utilização do sistema operativo *Linux*.

1.3 Objectivos

Este projecto tem como objectivo principal a implementação das normas IEEE 802.11p e IEEE 1609.4 em *Linux*, tirando partido das características do hardware previamente adquirido. Pretendemos, portanto, ter placas comunicantes na gama de frequência destinada às comunicações veiculares, com as principais características descritas por estas normas. Assim teremos dispositivos capazes de comunicar com qualquer dispositivo em qualquer canto do mundo, desde que estes respeitem as normas relacionadas.

De forma a poupar tempo na construção de toda a base necessária para a implementação deste projecto e visto que o hardware disponível apresenta condições para tal, decidiu-se que através de um *driver* específico para o hardware adquirido podemos atingir o principal objectivo deste projecto.

De forma a atingir o objectivo principal descrito acima, foi necessário o cumprimento dos seguintes objectivos:

- Implementação das alterações necessárias ao modo ad-hoc normal (criação de um novo modo wireless, como veremos mais à frente);
- Implementação das alterações à QoS especificadas na norma IEEE 802.11p;
- Implementação da funcionalidade de encaminhamento de canais, distinguindo os canais em CCH e SCH;
- Implementação da funcionalidade de comutação de canais, com suporte para o modo de acesso alternado;
- Implementação da comunicação com o receptor GPS de modo a garantir uma função de sincronização ao dispositivo;
- Implementação das primitivas descritas nas normas IEEE 802.11p/1609.4;

- Alteração da aplicação *iw* de modo a fornecer comandos de configuração/testes para algumas das funcionalidades implementadas.

O *driver*, denominado *ath5k*, é disponibilizado pelo grupo de desenvolvimento *Linux Wireless*, sendo um produto gratuito, o que consiste em uma de suas vantagens e satisfaz também outro objectivo proposto inicialmente por este projecto, consistindo na implementação do mesmo com o mínimo custo económico. Optamos, portanto, por aproveitar todas as características e funcionalidades disponibilizadas por este *driver*.

Capítulo 2

Estado da Arte

2.1 Introdução

O desenvolvimento deste projecto foi precedido de uma pesquisa de outros dispositivos disponíveis no mercado que apresentam alguma semelhança com o mesmo. Esta pesquisa teve como objectivo avaliar e fundamentar as escolhas efectuadas no decorrer do projecto e também permitir uma implementação tão actual quanto possível. Além dos dispositivos e diferentes projectos, também foi feito um levantamento da pilha protocolar que orienta a implementação das VANET's.

2.2 Protocolos

Como já foi referido anteriormente, o grupo de normas que descrevem a implementação das VANETs é constituido pela norma IEEE 802.11p e pela família de normas IEEE 1609.x. A publicação destas normas de reconhecimento mundial permite que haja homogeneidade na implementação por diferentes fabricantes, o que confere interoperabilidade entre diferentes dispositivos fabricados em várias partes do mundo.

A norma IEEE 802.11p define as alterações à norma IEEE 802.11 para acomodar as exigências impostas pela comunicação entre veículos. Esta norma especifica as características que um dispositivo WAVE deve apresentar de modo a operar na banda de frequência dos 5.9 GHz. Além de algumas especificações referentes à camada física, a norma IEEE 802.11p define algumas características a nível da sub-camada MAC incluindo a introdução do conceito da comunicação fora do contexto de uma *Basic Service Set* (BSS) [3]. A figura 2.1 ilustra a organização das várias normas que especificam a implementação das VANETs.

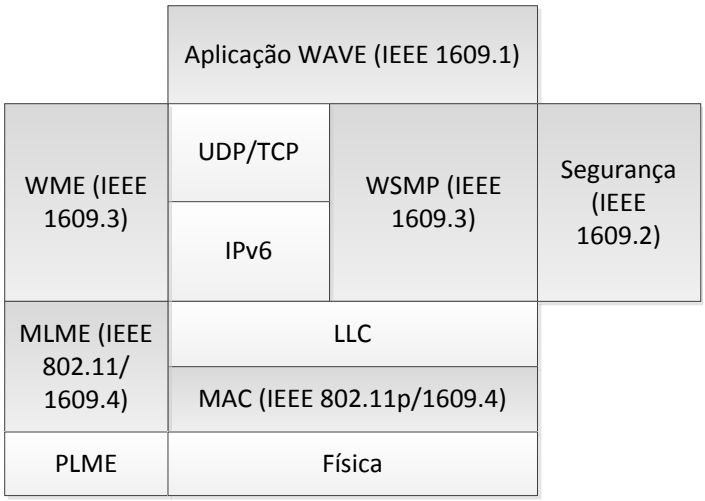


Figura 2.1: Protocolos por bloco da arquitectura WAVE

A família de normas IEEE 1609.x é formada pelas normas [8]:

- **IEEE 1609.0** - ***Architecture*** - Descreve a arquitectura WAVE e os serviços necessários à comunicação entre dispositivos WAVE/DSRC num ambiente veicular móvel (apenas foi lançado um *draft* até à data);

- **IEEE 1609.1** - ***Resource Manager*** - Especifica os serviços e interfaces da aplicação do gestor de recursos WAVE. Este descreve os serviços de dados e gestão oferecidos pela arquitectura WAVE. Define também formatos para mensagens de comando e as respostas apropriadas a estas mensagens, formatos para o armazenamento de dados que devem ser usados pelas aplicações para comunicar entre componentes da arquitectura, e formatos para mensagens de requisição e estado do sistema;

- **IEEE 1609.2** - ***Security Services for Applications and Management Messages*** - Define os formatos para mensagens de segurança e o processamento das mesmas. Esta norma também estabelece as circunstâncias em que há troca de mensagens de segurança e como estas mensagens devem ser processadas;

- **IEEE 1609.3** - ***Networking Services*** - Define os serviços da camada de rede e transporte, incluindo endereçamento e encaminhamento, para suportar a troca segura de

mensagens de dados WAVE. Esta norma define também as WSM, providenciando uma alternativa eficiente ao IPv6 que pode ser directamente suportado por aplicações;

- **IEEE 1609.4 - *Multi-channel Operation*** - Providencia melhoramentos à sub-camada MAC especificada pela norma IEEE 802.11 para suportar operações WAVE, nomeadamente a coordenação e encaminhamento de canais, mecanismos de *Quality of Service* (QoS) e sincronização dos dispositivos.

No que toca ao desenvolvimento deste projecto, as normas em foco são a IEEE 802.11p e a IEEE 1609.4, visto que estamos a lidar particularmente com as normas relacionadas com a sub-camada MAC. Foram publicadas em 2010 revisões destas duas normas pelo IEEE com algumas alterações importantes em relação às revisões anteriores, nomeadamente para a norma IEEE 1609.4. As características especificadas pelas normas anteriores serão discutidas e interpretadas no capítulo 3.

2.3 Produtos

A plataforma de teste destinada à avaliação de protocolos para comunicações veiculares denominada *LinkBird-MX* (figura 2.2), dispositivo fabricado pela NEC, possui as características principais apresentadas na tabela 2.1. A *LinkBird-MX* possui duas placas *wireless* mini-PCI, uma a operar no CCH e outra a operar num SCH. A operação multi-canal, a diferenciação de taxas de transferência, potências de transmissão e prioridades são controladas pelo software *C2X-SDK*, também disponibilizado pela NEC. Este software, apesar de actualizável, ainda suporta apenas o *draft* da norma IEEE 802.11p de Julho de 2007 [9].

Figura 2.2: LinkBird-MX (NEC) [10]

Tabela 2.1: Especificações técnicas da LinkBird-MX [9]

Parâmetro	**Detalhe**
Frequência (MHz)	5725 – 5925
Largura de Banda de Canal (MHz)	10, 20
Potência de Transmissão (dBm)	Max. 21
Taxas de Transmissão (Mbps)	6, 9, 12, 18, 24, 36, 48, 54 para 20 MHz de Largura de Banda 3, 4.5, 6, 9, 12, 18, 24, 27 para 10 MHz de Largura de Banda
Processador	Microprocessador MIPS de 64 bits a 266 MHz

A empresa Arada Systems disponibiliza dois dispositivos para DSRC, o *LocoMate RSU* (figura 2.3) e o *LocoMate OBU* (figura 2.4). O *LocoMate RSU* integra um receptor GPS com *Wi-Fi* e está apto a comunicar com um OBU na estrada ou com um outro RSU. O *LocoMate OBU*, por sua vez, integra um receptor GPS, *bluetooth*, CAN e *Wi-Fi*. Estes dois dispositivos formam a plataforma WAVE disponibilizada pela Arada Systems. A tabela 2.2 apresenta as especificações técnicas destes dispositivos [11]. Estes dispositivos suportam as normas IEEE 802.11p, *draft* lançado em 2007 [11], e as normas IEEE 1609.2, IEEE 1609.3 e IEEE 1609.4, sendo que estas, apesar de não ser explicitado no *datasheet* das placas, também constituem versões desactualizadas das mesmas. Os *LocoMates* utilizam *chipsets Wireless Local Area Network* (WLAN) da Atheros de versão AR5414 mini-PCI, equivalente aos utilizados na implementação do trabalho descrito nesta dissertação [12].

Figura 2.3: LocoMate RSU [13]

Figura 2.4: LocoMate OBU [14]

Tabela 2.2: Especificações Técnicas do LocoMate [11]

Parâmetro	**Detalhe**
Frequência (MHz)	5850 – 5925
Largura de Banda de Canal(MHz)	Canais 172 (5860 MHz) a 184 (5920 MHz) a 10 MHz, excepto 175 (5875 MHz) e 181 (5905 MHz) a 20 MHz
Protocolos	IEEE 802.11p (*draft*), IEEE 1609.2, IEEE 1609.3, IEEE 1609.4 (*draft*)
Taxas de Transmissão (Mbps)	6, 9, 12, 18, 24, 36, 48 para 20 MHz de Largura de Banda 3, 4.5, 6, 9, 12, 18, 24, 27 para 10 MHz de Largura de Banda
Potência de Transmissão (dBm)	Max. 21
Processador	680 MHz CPU
Operação Multi-canal	6.9 Mbps (Taxa de transmissão) Tempo de comutação de canais <= 3 ms

Tabela 2.3: Especificações Técnicas do eWAVE [15]

Parâmetro	**Detalhe**
Frequência (MHz)	2400 – 2484 MHz (ISM) 4940 – 4990 (PS) 5250 – 5350 (UNII) 5470 – 5725 (UNII) 5725 – 5825 (UNII) 5825 – 5850 (ISM) 5850 – 5925 (ITS-DSRC)
Protocolos	IEEE 802.11 a/b/g/j/p, IEEE 1609.2, IEEE 1609.3, IEEE 1609.4
Taxas de Transmissão (Mbps)	6, 9, 12, 18, 24, 36, 48 3, 4.5, 6, 9, 12, 18, 24, 27 1, 2, 5.5, 11
Potência de Transmissão (dBm)	Max. 14
Processador	32 bit MIPS 24K, 300 MHz

A Kapsch é uma empresa especializada principalmente no desenvolvimento de sistemas interoperáveis para implementações rodoviárias, como portagens inteligentes e comunicação entre veículos [16]. Esta desenvolveu um dispositivo denominado *eWAVE* (figura 2.5) e destina-se a oferecer capacidade de comunicação para aplicações usando 5.9 GHz DSRC. Este módulo contém, portanto, uma implementação da pilha de protocolos WAVE.

Além do *eWAVE* a Kapsch disponibiliza o MCU, *Mobile Communication Unit*, e o *WAVE starter kit*. O *WAVE Starter Kit* é uma plataforma de hardware e software para o desenvolvimento de aplicações utilizando 5.9 GHz DSRC. O MCU apresenta-se como a solução *On-Board* da Kapsch, construída a partir do *eWAVE*. As aplicações desenvolvidas a partir do *WAVE Starter Kit* podem ser instaladas e testadas na estrada a partir do MCU [15].

Figura 2.5: eWAVE [17]

Figura 2.6: MCNU [18]

Figura 2.7: *Wireless Starter Kit* [15]

A Kapsch desenvolveu também uma solução RSU denominada *Multiband Configurable Networking Unit* (MCNU). Este dispositivo suporta os protocolos exigidos à comunicação 5.9 GHz DSRC [19]. O MCNU e MCU formam a plataforma 5.9 GHz DSRC desenvolvida pela Kapsch.

Tabela 2.4: Especificações Técnicas do MCNU [19]

Parâmetro	**Detalhe**
Frequência (MHz)	2400 – 2484 (ISM) 4940 – 4990 (PS) 5150 – 5250 (UNII) 5250 – 5350 (UNII) 5470 – 5725 (UNII) 5725 – 5825 (UNII) 5825 – 5850 (ISM) 5850 – 5925 (ITS-DSRC)
Protocolos	IEEE 802.11 a/b/g/j/p, IEEE 1609.2, IEEE 1609.3, IEEE 1609.4
Taxas de Transmissão (Mbps)	6, 9, 12, 18, 24, 36, 48 3, 4.5, 6, 9, 12, 18, 24, 27 1, 2, 5.5, 11
Processador	Pentium grade (1GHz)

A CohdaWireless desenvolveu o dispositivo *Cohda MK2 WAVE-DSRC Radio* que é uma solução completa da norma IEEE 802.11p RF/PHY/MAC e permite actualização do software para implementação das normas IEEE 1609.x e também das normas IEEE 802.11a/g, disponibilizando um receptor GPS embutido e um processador que suporta o *Linux* [20].

A DENSO desenvolveu a plataforma de hardware *Wireless Safety Unit* (WSU) projectado para acomodar os requisitos apresentados no *draft* de 2007 da norma IEEE 802.11p [21].

A Unex desenvolveu a placa mini-PCI *wifi* denominada *DCMA-86P2* (figura 2.8), cujas especificações tecnicas são apresentadas na tabela 2.5. Esta placa apresenta um baixo custo e acomoda as especificações necessárias à implementação das normas IEEE 802.11p e as normas WAVE. Como podemos ver na tabela 2.5, a *DCMA-86P2* contém um *chipset* desenvolvido pela *Atheros*, cuja versão é suportada pelo driver *ath5k*.

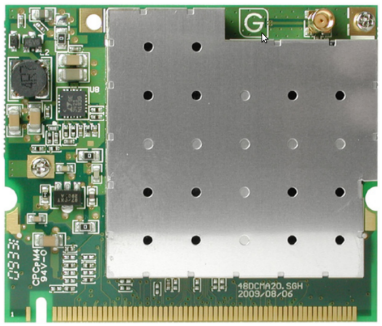


Figura 2.8: DCMA-86P2 [22]

Tabela 2.5: Especificações Técnicas da DCMA-86P2 [22]

Parâmetro	**Detalhe**
Chipset	Atheros AR5414A-B2B
Frequência (MHz)	5860 – 5925
Largura de Banda de Canal(MHz)	5, 10, 20, 40
Potência de Transmissão (dBm)	2.5 – 24
Taxas de Transmissão (Mbps)	6, 9, 12, 18, 24, 36, 48, 54 a 20 MHz 3, 4.5, 6, 9, 12, 18, 24, 27 a 10 MHz 1.5, 2.75, 3, 4.5, 6, 9, 12, 13.5 a 5 MHz
Antena	DIP MMCX RF (conector)

O dispositivo utilizado no trabalho descrito por esta dissertação, corresponde ao dispositivo wireless desenvolvido pela Unex, *DCMA-86P2*. O conjunto de hardware comprado para o desenvolvimento deste trabalho ronda os 300 euros, o que é favorável para o projecto DRIVE-IN, mantendo o objectivo económico do mesmo. Em relação às outras placas aqui referidas, esta é muito mais económica, visto que o valor de mercado destas está entre os 1200 e os 3000 euros.

2.4 Projectos

O projecto SAFESPOT [23] é um projecto de investigação europeu que implementa uma rede de comunicação cooperante V2V e V2I, respeitando a norma IEEE 802.11p [24], versão disponibilizada até 2008. Este projecto foi implementado a partir do *driver Madwifi*, *driver* que deu origem ao *ath5k* utilizado no projecto DRIVE-IN.

O grupo de investigação e tecnologia da BMW em conjunto com a DENSO, apresentam o projecto *OpenWAVE Engine/WSU*. Este projecto utiliza o WSU disponibilizado pela DENSO. O *OpenWAVE Engine* foi inicialmente denominado por *ACUp* (*AKTIV Communication Unit - 802.11p*) desenvolvido pela BMW para o projecto de investigação alemão *AKTIV*. O *OpenWAVE Engine* implementa a parte do *software* WAVE que é perfeitamente compatível com o DENSO WSU. Os protocolos da família IEEE 1609.x implementados são os referentes às revisões até 2007 [21].

O projecto CABERNET foi um projecto realizado em Boston, onde 25 taxis locais foram equipados de forma a compreender a troca de informação entre dispositivos. Este projecto foi realizado em 2008, e a sua implementação utilizou para tal as normas IEEE 802.11b/g [25].

Em 2007, o projecto SUVnet equipou 4000 taxis locais em Shanghai, com GPS e dispositivos utilizando uma norma IEEE 802.11, cuja versão não foi especificada [26].

CarTel é uma *testbed* realizada em Seattle, onde 6 veículos foram equipados com dispositivos contendo vários sensores, bluetooth e placas IEEE 802.11b. Esta *testbed* foi realizada durante um ano, recolhendo continuamente informação acerca da comunicação *Wi-Fi* numa área metropolitana [27].

DRIVE é uma *testbed* realizada pela Telefonica que visa avaliar os vários tipos de serviços para VANET. Este projecto continua a respeitar apenas as normas IEEE 802.11b/g [28].

Por fim, o projecto DRIVE-IN consiste num projecto de desenvolvimento de software base para VANET, com o objectivo da realização de uma *testbed* onde 465 taxis e alguns auto-carros locais da cidade do Porto serão equipados [29]. A realização de uma *testbed* desta dimensão contribuirá para uma análise eficiente, actual e credível do comportamento de um sistema WAVE implementado em situações reais. Esta implementação possibilita o desenvolvimento e teste de aplicações WAVE ou algoritmos de *routing*, por exemplo, por outros projectos, visto que, ao contrário de projectos comerciais, todo o sistema implementado é de livre acesso. O projecto DRIVE-IN apresenta-se, portanto, como uma mais valia para a investigação na área das comunicações veiculares.

Capítulo 3

A Sub-Camada MAC IEEE 802.11p/1609.4

Neste capítulo serão descritas as funcionalidades introduzidas pelas normas IEEE 1609.4 e IEEE 802.11p. Todas as funcionalidades descritas neste capítulo têm como referências [3] e [5].

3.1 Introdução

A norma IEEE 1609.4 apresenta-se como uma extensão das normas IEEE 802.11 e IEEE 802.11p anteriormente publicadas, descrevendo modificações necessárias à implementação da sub-camada MAC para ambientes veiculares. A norma IEEE 802.11p especifica algumas alterações à norma IEEE 802.11 relativas à QoS, alterações à camada física e a introdução do conceito da comunicação fora do contexto de uma BSS. Este conceito descreve o CCH definido na norma IEEE 1609.4, um dos conceitos mais importantes no âmbito das comunicações veiculares. O CCH corresponde a um canal, número 178, correspondente à frequência 5.89 GHz, que funciona como um canal fixo ao qual qualquer dispositivo WAVE pode aceder e comunicar sem a necessidade de haver algum tipo de associação ou autenticação para tal.

Como podemos ver na figura 3.1, a sub-camada MAC apresenta uma entidade de gestão chamada *MAC subLayer Management Entity* (MLME). Esta entidade funciona como uma interface que monitora o estado da sub-camada MAC e também interfere em termos de decisão quando há comunicação com a sub-camada LLC ou a camada física. Cada camada apresenta uma entidade gestora, como a camada física apresenta a *PHY subLayer Management Entity* (PLME), sendo que a entidade gestora da sub-camada MAC é definida na norma IEEE 802.11,

mas foi necessário uma extensão da mesma de modo a suportar as alterações introduzidas na passagem para a sub-camada MAC WAVE.

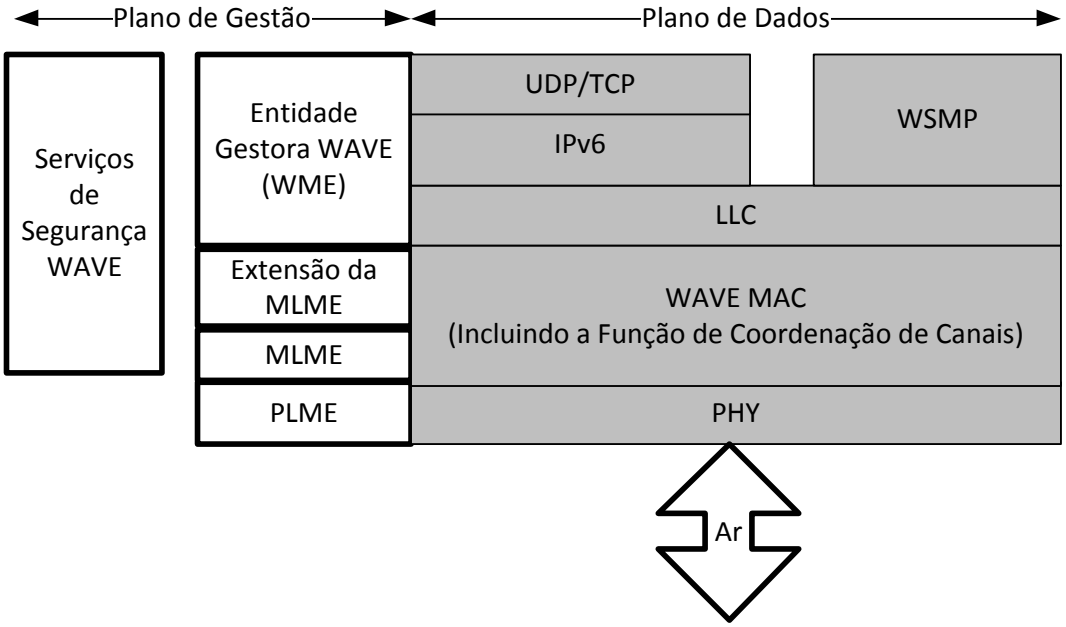


Figura 3.1: Modelo de Referência

A norma IEEE 1609.4 apresenta os serviços de gestão e controlo de canais que a sub-camada MAC deve providenciar. Estes serviços são apresentados a seguir. O plano de serviços destinados a dados é o seguinte:

- Coordenação de canais (*Channel Coordination*): A sub-camada MAC coordena os intervalos de canais para que os pacotes de dados sejam transmitidos no canal *Radio Frequency* (RF) no instante correcto.

- Encaminhamento de canais (*Channel Routing*): A sub-camada MAC trata os dados que são enviados ou recebidos de uma camada superior. Esta especificação inclui encaminhamento de pacotes de dados da sub-camada LLC para o canal designado, e altera os parâmetros para transmissões WAVE.

- Prioridade do utilizador (*User Priority*): A norma WAVE suporta uma variedade de aplicações seguras e não seguras com até 8 níveis de prioridade como definido na norma IEEE 802.11. O uso da prioridade do utilizador, *User Priority* (UP), e relativa categoria

de acesso, *Access Category* (AC), confere QoS usando o mecanismo EDCA, especificado na norma IEEE 802.11.

O plano de serviços destinados à gestão é o seguinte:

- Sincronização multi-canal (*Multi-Channel Sincronization*): A MLME utiliza informação trocada localmente e recebida pelo ar para providenciar uma função de sincronização com o objectivo de alinhar os intervalos dos canais entre os dispositivos WAVE comunicantes. A MLME providencia a capacidade de gerar tramas de aviso temporal, *Timing Advertisement frame* (TA), para distribuir informação ao sistema, e monitora tramas TA recebidas.

- Acesso ao canal (*Channel Access*): A MLME controla o acesso a canais de rádio específicos em resposta à requisição de comunicação recebidas pela *WAVE Management Entity* (WME).

- *Vendor Specific Action frame* (VSA): A MLME aceita tramas VSA e passa-las-à para a WME. A MLME também gerará tramas VSA para transmissão quando requisitado pela WME.

Note-se que nesta dissertação apenas serão tratados os serviços implementados por este trabalho, sendo que as tramas VSA e TA não serão abordadas, pois constituem parte do trabalho futuro.

3.2 Arquitectura

A figura 3.2 apresenta o exemplo contido na norma IEEE 1609.4 para a arquitectura interna da sub-camada MAC WAVE com operação multi-canal para o caminho de transmissão. Na mesma podemos observar as operações de transmissão encaminhamento de canais, enfileiramento de pacotes, priorização e coordenação de canais.

Existem dois tipos de canais especificados nesta arquitectura, o canal de controlo (CCH) e o canal de serviço (SCH). O CCH permite a um dispositivo WAVE a troca de mensagens WSM sem a necessidade de uma pré-associação, possibilitando assim a recepção e o envio de alertas das condições que o utilizador irá encontrar na estrada de forma instantânea. Por razões de segurança e congestionamento no envio de pacotes, o CCH pode enviar qualquer tipo de tramas, excepto dados IP. Um canal SCH deve enviar e receber dados IP ou WSM.

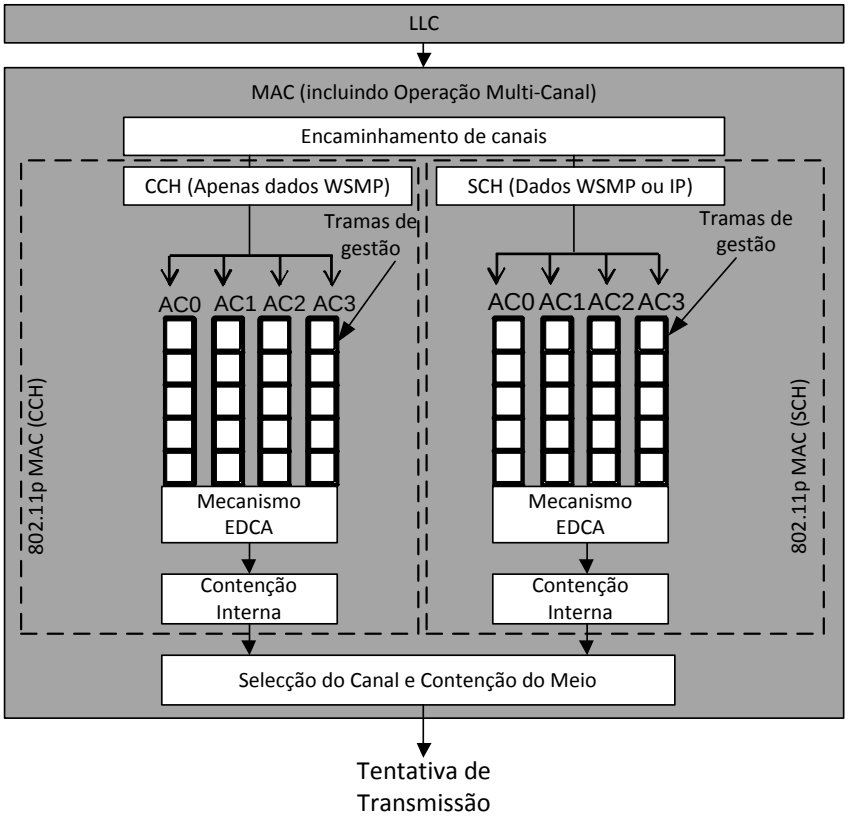


Figura 3.2: Arquitectura interna da sub-camada MAC WAVE multi-canal

3.3 Coordenação e Acesso a Canais

A coordenação de canais foi desenvolvida com vista em permitir um ou mais dispositivos o acesso ao CCH e ao SCH em intervalos bem definidos, de forma a comunicarem os seus serviços, ou vislumbrarem outros serviços disponíveis, no CCH e continuarem a servir ou serem servidos no SCH. Desta forma, é necessária uma operação que coordene a comutação entre os canais, assegurando que quando os mesmos voltarem a aceder ao canal em questão num intervalo de tempo futuro, as configurações do mesmo não sejam perdidas, assim como a continuação da tarefa que estava a decorrer, e que foi suspensa devido ao término do intervalo para o canal, seja retomada. Esta operação não é sempre necessária, pois existem diferentes formas de acesso ao canal e nem todas exigem coordenação.

A norma IEEE 1609.4 apresenta quatro formas possíveis de acesso ao canal (figura 3.3): acesso contínuo (*Continuous Access*), acesso alternado ao canal de serviço (*Alternating Service*

Channel Access), acesso imediato ao canal de serviço (*Immediate Service Channel Access*) e acesso extendido ao canal de serviço (*Extended Service Channel Access*).

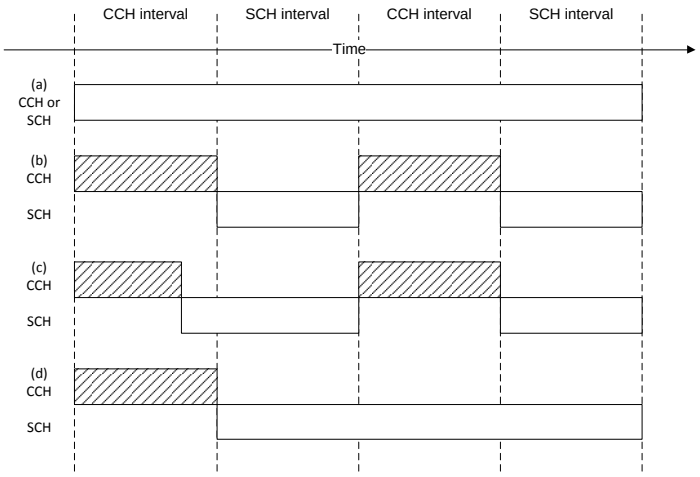


Figura 3.3: Formas de Acesso ao Canal: (a) contínuo, (b) alternado, (c) imediato e (d) extendido

O acesso contínuo apresenta-se como o mais simples das quatro formas de acesso, sendo destinado a dispositivos WAVE que apenas queiram monitorar o CCH, ou ter um acesso contínuo a um serviço disponível num SCH previamente conhecido. Qualquer dispositivo WAVE permanece no CCH com acesso contínuo inicialmente e é feito um pedido à sub-camada MAC, quando necessário, para aceder a um SCH à partir da primitiva *MLMEX-SCHSTART.request*, descrita na secção 3.7.1, indicando o acesso pretendido.

O acesso alternado apresenta-se como um desafio em termos de implementação, pois este implica a salvaguarda de toda a configuração do canal que está prestes a ser desactivado, assim como devem ser postas em espera todas as filas de transmissão e recepção de dados, para quando voltarmos ao mesmo possamos ter todas as configurações repostas e continuarmos as tarefas que foram suspensas e que agora serão retomadas. Em termos de procedimento da sub-camada MAC, a MLME deverá enviar um pedido à PLME no início de cada intervalo do CCH e do SCH, no intervalo de guarda (*Guard Interval*), pedindo a mudança para o canal respeitante ao intervalo que se inicia.

O acesso imediato ao SCH constitui uma forma de acesso ao canal opcional em que o dispositivo acede ao SCH pretendido sem a necessidade de esperar pelo início de um intervalo do SCH. Para requerer um acesso imediato, a *flag ImmediateAccess*, um dos campos da

primitiva *MLMEX-SCHSTART.request*, deve ser posta a *True*. Assim que possível a MLME envia um pedido à PLME indicando a mudança de canal para o canal pretendido.

O acesso extendido ao SCH também se apresenta como uma forma opcional de acesso ao canal em que é permitida a comunicação num SCH sem a necessidade de voltar ao CCH, durante um certo número de intervalos de tempo. A MLME envia um pedido à PLME no início de um intervalo do SCH para a mudança de canal para o SCH pretendido. Assim, a MLME não envia qualquer pedido de mudança de canal à PLME durante um certo número de intervalos de tempo, especificado antes por um dos campos da primitiva *MLMEX-SCHSTART.request* denominado *ExtendedAccess* (sendo este um número inteiro maior do que zero). Quando o número de intervalos de tempo especificado pelo campo anterior chegar ao fim, a MLME retorna ao acesso contínuo ou alternado.

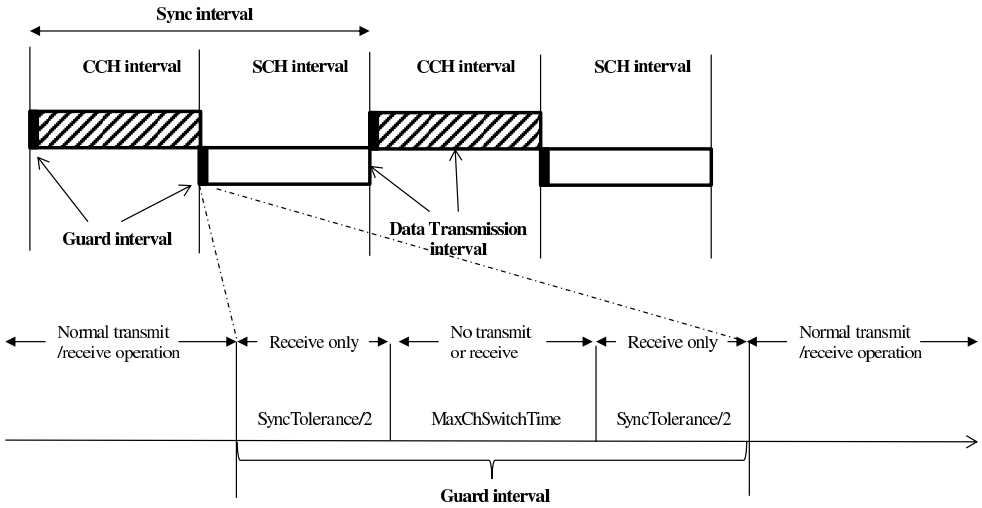


Figura 3.4: Diferentes intervalos de tempo (acesso alternado) e decomposição do intervalo de guarda

Como vimos anteriormente, a implementação das diferentes formas de acesso aos canais implica a introdução de intervalos de tempo bem definidos, os quais todos os dispositivos WAVE comunicantes devem respeitar. A figura 3.4 mostra o exemplo de acesso alternado com os diferentes intervalos de tempo. Note-se que um intervalo de sincronização (*Sync Interval*) corresponde à soma de um intervalo do CCH (*CCH Interval*) e um intervalo do SCH (*SCH Interval*). Também se nota a partir da figura que no início de cada intervalo temos um breve intervalo de tempo denominado de intervalo de guarda (*Guard Interval*). É de salientar que

um intervalo de sincronização deve iniciar-se juntamente com o início do segundo *Universal Time Coordinated* (UTC), sendo esta a base temporal comum para a coordenação de canais WAVE.

O intervalo de guarda aparece na norma IEEE 1609.4 como o intervalo de tempo necessário para que todos os dispositivos WAVE troquem de canal e analisem o estado da sincronização temporal. A figura 3.4 mostra também a decomposição do intervalo de guarda para melhor compreensão do mesmo.

Podemos notar na figura 3.4 os intervalos de tolerância (*Sync Tolerance*) e de tempo máximo para troca de canais (*MaxChSwitchTime*). É a partir de metade de um intervalo de tolerância que um dispositivo WAVE analisa se está ou não sincronizado com o UTC. O intervalo *MaxChSwitchTime* corresponde ao tempo máximo que todos os dispositivos WAVE têm disponível para trocarem de canal. Um intervalo *MaxChSwitchTime* representa o intervalo de tempo em que um dispositivo WAVE estará inactivo, não havendo transmissões ou recepções neste intervalo.

Durante um intervalo de guarda, o dispositivo WAVE estará em estado de transição, logo não poderá transmitir qualquer tipo de trama para outros dispositivos. Sendo assim, quando um intervalo de guarda se inicia, a sub-camada MAC deve suspender todas as tarefas de transmissão, guardando a informação acerca do canal, e efectua a troca de canais, iniciando a comunicação num novo canal ou retomando a comunicação num canal anteriormente suspenso.

3.4 Encaminhamento de Canais

Um dispositivo WAVE deve suportar tráfego WSMP, IP ou ambos, sendo que é necessário haver um mecanismo de encaminhamento de canais para a transmissão e recepção dos diferentes pacotes no canal pretendido. Como já vimos na secção 3.2, o CCH não suporta dados IP, sendo também um dos objectivos do mecanismo de encaminhamento de canais o bloqueio deste tipo de tramas no CCH. Quando uma trama é passada para a sub-camada MAC, através de primitivas enviadas pela sub-camada LLC, esta apresenta vários campos que identificam o número do canal, a potência de transmissão ou a prioridade da trama, por exemplo. O campo *Ethertype* indica qual o protocolo a que a trama respeita, sendo utilizado por camadas superiores para informar a sub-camada MAC qual o protocolo em questão. A partir deste campo informativo pode-se facilmente bloquear a passagem de dados IP no CCH.

A transmissão e recepção de tramas WSMP e IP são requisitadas através de primitivas que contêm informação acerca dos parâmetros que devem ser utilizados na mesma. Este

mecanismo permite às camadas superiores controlar os parâmetros da *Physical Layer* (PHY) para cada pacote.

Figura 3.5: MA-UNITDATAX.req para dados WSMP

Nota-se da figura 3.5 que uma trama de dados WSMP é passada com informação acerca do número do canal, potência de transmissão e taxa de transferência de dados, enquanto que uma trama de dados IP (figura 3.6) necessita de um identificador, o perfil do transmissor (*Transmitter Profile*).

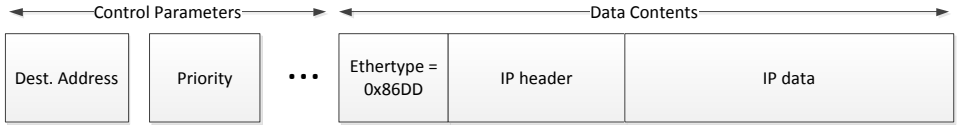


Figura 3.6: MA-UNITDATA.req para dados IP

Antes de transmitir dados IP, a MLME obriga ao registo de um perfil do transmissor, contendo informação sobre o número do SCH, potência de transmissão e taxa de transferência de dados, sendo guardada esta informação numa tabela de perfis (*TransmitterProfileTable*) na *Management Information Base* (MIB), com um identificador. A partir deste a MLME acede aos diferentes parâmetros associados para proceder à transmissão para o SCH em questão. Se o perfil não estiver registado a MLME deve descartar o pacote em questão.

3.5 Sincronização Temporal

Para suportar a operação de coordenação de canais, a sub-camada MAC deve manter uma função destinada à sincronização dos diferentes dispositivos WAVE, mantendo assim uma base temporal comum de referência. Este mecanismo permite a dispositivos que não possuem uma fonte temporal local, sincronizarem-se com outros dispositivos através da recepção de

mensagens com informação temporal, provenientes de outros dispositivos com capacidade para tal.

A função de sincronização utiliza o UTC como base temporal comum, que pode ser obtido através de um dispositivo receptor GPS. Visto que um dispositivo receptor GPS providencia um sinal UTC de 1 *Pulse Per Second* (PPS), com erro inferior a 100 ns, a MLME pode utilizar este pulso para determinar o instante inicial de um intervalo de canal, visto que no início de um segundo começa um intervalo de canal, sincronizando os diferentes dispositivos a partir da recepção deste pulso.

Os relógios internos dos dispositivos WAVE apresentam desvios ao longo do tempo, o que pode ser estabilizado a partir da utilização de um dispositivo receptor GPS e de um filtro de *Kalman*, sugerido pela norma IEEE 1609.4. O filtro de *Kalman* é utilizado para estimar o tempo UTC a partir do relógio interno do dispositivo e do pulso obtido pelo receptor GPS.

3.6 Qualidade de Serviço

O mecanismo EDCA é utilizado pela sub-camada MAC WAVE no acesso prioritizado para transmissão de dados. Este mecanismo está especificado na norma IEEE 802.11, sendo o mesmo utilizado pela norma IEEE 1609.4, mas com diferentes valores para os parâmetros EDCA. As alterações aos valores dos parâmetros EDCA para dispositivos WAVE, estão especificados em IEEE 802.11p.

O mecanismo EDCA oferece serviços de QoS priorizados que corresponde todo o tráfico destinado à sub-camada MAC para a AC correspondente e diferencia a probabilidade de conseguir uma oportunidade de transmissão, *Transmit Opportunity* (TXOP), usando diferentes parâmetros de acesso ao canal, denominados parâmetros EDCA [30].

Quando uma primitiva *MA-UNITDATA.request* ou *MA-UNITDATAX.request* é passada à sub-camada MAC, é efectuado o encaminhamento de canais e de seguida a sub-camada MAC deve enviar os dados para a fila correspondente à categoria de acesso, AC, identificada pela prioridade de utilizador, UP, que constitui uma das informações presentes na primitiva recebida pela sub-camada MAC, como se pode observar nas figuras 3.5 e 3.6. Como especificado na norma IEEE 802.11, os valores de UP são correspondidos às ACs como mostra a tabela 3.1.

Existem quatro ACs, numa gama de 1 a 4, com índices de 0 a 3, em que 4 constitui a mais alta prioridade. As tramas de gestão são marcadas com a prioridade mais alta, dada por *Access Category Index* (ACI) de índice 3 (figura 3.2), como especifica a norma IEEE 802.11.

De seguida apresentam-se os parâmetros EDCA e uma breve definição dos mesmos:

- *Arbitration Inter-Frame Space* (AIFS): Intervalo mínimo temporal entre o instante em que o meio *wireless* fica inactivo e o instante em que a transmissão de uma trama se inicia;

- *Contention Window* (CW): Um intervalo de contenção da janela temporal, escolhido aleatoriamente entre os valores mínimo e máximo, previamente estabelecidos, utilizado na implementação do mecanismo *back-off* de transmissão;

- TXOP limit: O intervalo de tempo máximo que uma estação pode transmitir após obter uma TXOP. O valor 0 para este limite permite apenas a transmissão de um único *MAC Protocol Data Unit* (MPDU).

Tabela 3.1: UP para AC

Prioridade	**UP**	**Designação 802.1D**	**AC**	**Designação**
Baixa	1	BK	AC_BK	Background
	2	-	AC_BK	Background
	0	BE	AC_BE	Best Effort
	3	EE	AC_BE	Best Effort
	4	CL	AC_VI	Vídeo
	5	VI	AC_VI	Vídeo
	6	VO	AC_VO	Voz
Alta	7	NC	AC_VO	Voz

A tabela 3.2 apresenta os valores dos parâmetros EDCA que devem ser utilizados pela sub-camada MAC como valores padrão utilizados no CCH, ou para SCHs que não tenham especificado os valores dos parâmetros EDCA aquando da sua inicialização, através da primitiva *MLMEX-SCHSTART.request*, descrita na secção 3.7.1. Estes valores são especificados na norma IEEE 802.11p como valores padrão para operações fora do contexto de uma BSS. O algoritmo interno de contenção de acordo com a norma IEEE 802.11, calcula o *back-off* independentemente para cada AC baseado nos parâmetros EDCA. Assim, a AC de menor *back-off* "ganha"a contenção interna e acede ao meio *wireless*. Os valores para *aCWmin* e *aCWmax* são 15 e 1023 *slot times*, respectivamente.

Tabela 3.2: Parâmetros EDCA padrão (IEEE 802.11p)

AC	**CWmin**	**CWmax**	**AIFSN**	**TXOP**
AC_BK	$aCWmin$	$aCWmax$	9	0
AC_BE	$aCWmin$	$aCWmax$	6	0
AC_VI	($aCWmin$+1)/2 - 1	$aCWmin$	3	0
AC_VO	($aCWmin$+1)/4 - 1	($aCWmin$+1)/2 - 1	2	0

O AIFS é calculado através do *Arbitration Inter-Frame Space Number* (AIFSN) definido para cada AC, o SIFS, tempo mínimo de espera entre o envio de duas tramas, e o *slot time*:

AIFS[AC] = AIFSN[AC] x SIFS x *slot time*

O CW é calculado consoante os valores predefinidos apresentados na tabela 3.2. O valor configurado inicialmente corresponde a $aCWmin$. Se houver uma falha na transmissão da trama, o valor de CW[AC] passa a ser o seguinte, tendo em conta que o limite máximo é dado por $aCWmax$:

CW[AC] = (CW[AC] + 1) x 2 - 1

Se a transmissão da trama for feita com sucesso, o valor de CW[AC] permanece igual a $aCWmin$. Assim, o tempo de *backoff* é escolhido de forma aleatória, por um valor na gama [0,CW[AC]], multiplicado pelo *slot time*. A figura 3.7 apresenta um exemplo da diferença entre os tempos de espera de cada AC, onde AIFS e o tempo de *backoff* são representados.

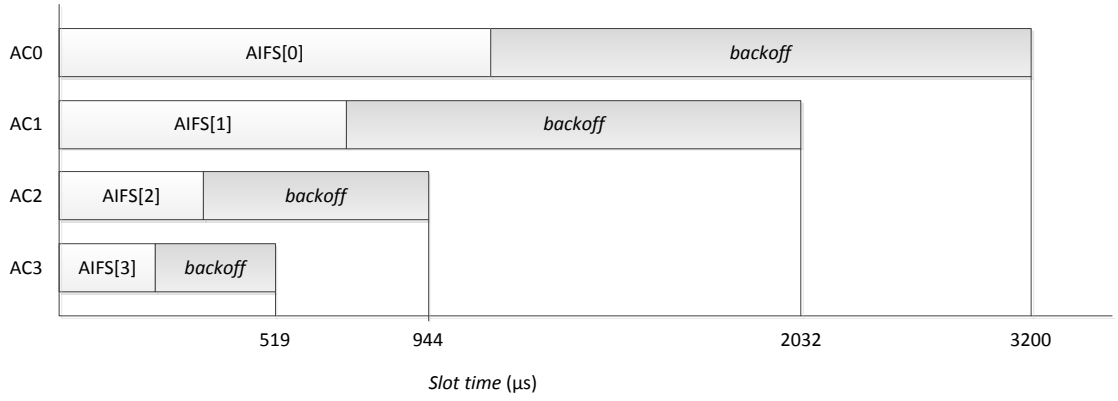


Figura 3.7: Exemplo dos diferentes tempos de espera possíveis para cada AC. Os valores apresentados foram calculados assumindo que SIFS e um *slot time* correspondem a 16 micro-segundos

3.7 Primitivas

A norma IEEE 1609.4 define a extensão da MLME descrita na norma IEEE 802.11, denominada *MAC subLayer Management Entity Extension* (MLMEX). A tabela 3.3 apresenta o sumário das primitivas implementadas no trabalho descrito nesta dissertação. Todas as primitivas implementadas estão especificadas na norma IEEE 1609.4. De seguida, será feita uma breve descrição de cada primitiva.

As seguintes secções podem apresentar três tipos de primitivas:

- Requisição por parte da WME algum serviço ou informação à MLME;
- Confirmação por parte da MLME à WME indicando o resultado da requisição feita;
- Indicação por parte da MLME indicando à WME alguma acção inesperada, que pode ser uma anomalia ou a recepção de alguma trama.

Tabela 3.3: Sumário de primitivas

SAP	Primitiva	Especificada em
MLMEX	MLMEX-SCHSTART	IEEE 1609.4
	MLMEX-SCHEND	IEEE 1609.4
	MLMEX-REGISTERTXPROFILE	IEEE 1609.4
	MLMEX-DELETETXPROFILE	IEEE 1609.4
	MLMEX-CANCELTX	IEEE 1609.4

3.7.1 MLMEX-SCHSTART

A primitiva *MLMEX-SCHSTART.request* é gerada pela WME para requisitar à MLME o acesso a um SCH. Esta primitiva fornece os seguintes campos à MLME:

- *Channel Identifier*: Número do canal onde a WME requisita o acesso;
- *OperationalRateSet*: Número entre 2 e 127 que corresponde ao valor da taxa de transferência a ser utilizada, em incrementos de 500 kbps. Sendo assim, este valor indica uma taxa de transferência de 1 a 63.5 Mbps [4];
- *EDCA Parameter Set*: Este campo indica os valores dos parâmetros EDCA para diferentes tipos de prioridade;

- *ImmediateAccess*: Variável booleana que indica se a função de coordenação de canais deve esperar até ao próximo intervalo do SCH ou deverá realizar uma troca de canal imediata;
- *ExtendedAccess*: Variável que indica quantos intervalos do SCH o acesso permanecerá contínuo, mudando para o CCH no final de *ExtendedAccess* intervalos do SCH.

Depois da recepção desta primitiva, a MLME inicializa a função de coordenação de canais para o SCH indicado por *Channel Identifier*. Note que se o serviço a ser disponibilizado neste SCH utilizar dados IP, o acesso ao SCH indicado só é concedido se um perfil de transmissor para este canal já estiver registado na tabela de perfis de transmissor, ou seja, implica que a primitiva *MLMEX-REGISTERTXPROFILE.request* já tenha sido invocada anteriormente para este SCH.

A primitiva *MLMEX-SCHSTART.confirm* é gerada pela MLME de modo a confirmar à WME se o pedido de acesso ao SCH foi aceite ou não. Esta primitiva possui apenas um campo, denominado *ResultCode* que indica se o pedido foi aceite (*SUCCESS*), se foi parcialmente aceite (*PARTIAL_SUCCESS*) devido ao facto de o dispositivo WAVE não suportar acesso imediato, extendido ou ambos, se o dispositivo não se encontra sincronizado (*NO_SYNC*) ou se houve uma falha por outro motivo não especificado (*UNSPECIFIED_FAILURE*). A figura 3.8 ilustra a interacção entre WME e a MLME a partir das primitivas aqui descritas.

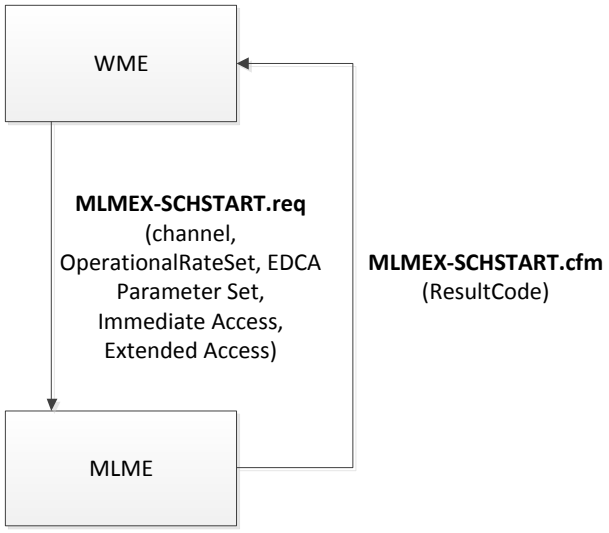


Figura 3.8: Requisição do acesso a um SCH

3.7.2 MLMEX-SCHEND

A primitiva *MLMEX-SCHEND.request* é gerada pela WME para requisitar à MLME o término do acesso a um SCH. Esta primitiva fornece o campo *Channel Identifier* à MLME. A MLME termina o acesso ao SCH indicado por *Channel Identifier*, invocando a primitiva *MLMEX-DELETETXPROFILE.request* se o serviço utiliza dados IP, de modo a apagar o registo deste SCH da tabela de perfis de transmissor.

A primitiva *MLMEX-SCHEND.confirm* é gerada pela MLME em resposta da recepção de uma primitiva *MLMEX-SCHEND.request*, indicando à WME através do campo *ResultCode*, se o acesso foi terminado com sucesso (*SUCCESS*) ou se o número do SCH indicado em *Channel Identifier* é inválido (*INVALID_PARAMETERS*). A figura 3.9 ilustra a interacção entre WME e a MLME quando a primitiva *MLMEX-SCHEND.request* é gerada.

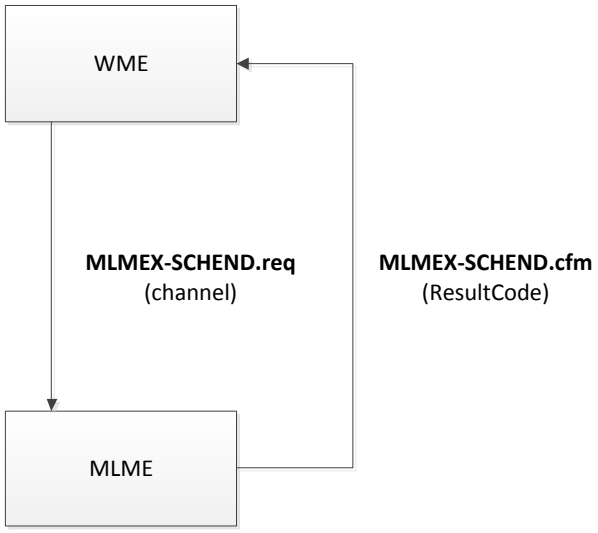


Figura 3.9: Requisição do término do acesso a um SCH

A primitiva *MLMEX-SCHEND.indication* é gerada pela MLME para indicar à WME que o acesso a um SCH foi terminado por parte da MLME sem a recepção da primitiva *MLMEX-SCHEND.request*. Esta primitiva fornece o campo *Reason* à WME, onde indica a razão do término do acesso ao SCH, que pode ser devido a perda de sincronização por parte do dispositivo WAVE (*LOSS_OF_SYNC*) ou devido a outras razões não especificadas (*UNSPECIFIED_REASON*). A figura 3.10 ilustra a interacção entre WME e a MLME quando

a primitiva *MLMEX-SCHEND.indication* é gerada.

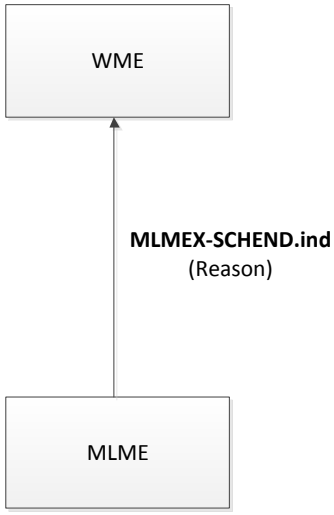


Figura 3.10: Indicação do término do acesso a um SCH

3.7.3 MLMEX-REGISTERTXPROFILE

A primitiva *MLMEX-REGISTERTXPROFILE.request* é gerada pela WME pedindo o registo à MLME de um perfil de transmissor para um SCH onde serão transmitidos dados IP. Esta primitiva fornece os seguintes campos à MLME:

- *Channel Identifier*: Número do SCH onde se pretende transmitir dados IP;
- *Adaptable*: Variável booleana que indica se os valores passados nos campos *TxPwr_Level* e *DataRate* são adaptaveis ou não. Se este valor corresponder a *TRUE*, este indica o valor máximo a que estes dois valores podem chegar, mas sendo que podem variar entre os valores abaixo dos mesmos. Se este valor corresponder a *FALSE*, este indica que os valores passados nos dois campos referidos são fixos;
- *TxPwr_Level*: Índice do nível de potência de transmissão da tabela *dot11PhyTxPowerTable* que faz parte da MIB da camada física;
- *DataRate*: Índice da taxa de transmissão guardada na MIB da camada física;

A primitiva *MLMEX-REGISTERTXPROFILE.confirm* é gerada pela MLME de modo a indicar à WME o resultado do pedido efectuado com a recepção da primitiva *MLMEX-*

REGISTERTXPROFILE.request. O campo *ResultCode* fornecido pela primitiva *MLMEX-REGISTERTXPROFILE.confirm*, indica se o perfil do transmissor foi registado com sucesso (*SUCCESS*) ou se algum dos parâmetros passados foi considerado inválido (*INVALID_PARAMETERS*). A figura 3.11 ilustra a interacção entre WME e a MLME quando a primitiva *MLMEX-REGISTERTXPROFILE.request* é gerada.

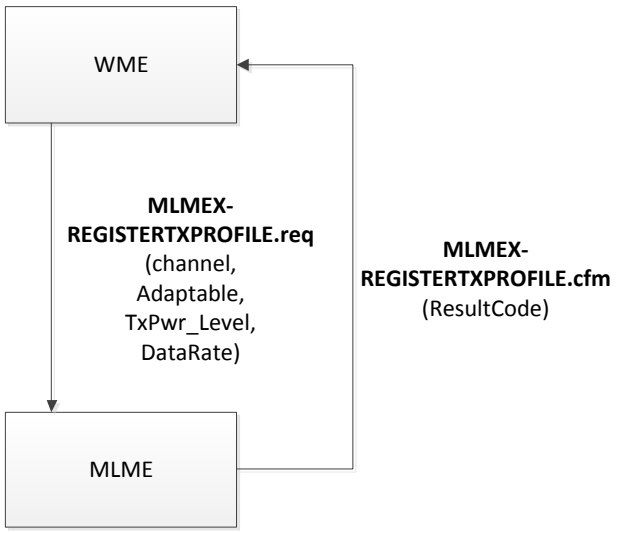


Figura 3.11: Requisição do registo de um perfil de transmissor para um SCH

3.7.4 MLMEX-DELETETXPROFILE

A primitiva *MLMEX-DELETETXPROFILE.request* é gerada pela WME pedindo à MLME a eliminação de um perfil de transmissor de um SCH indicado pelo campo *Channel Identifier*. Depois da recepção desta primitiva, a MLME acede à tabela de perfis de transmissor e apaga a entrada referente ao canal indicado por *Channel Identifier*.

A primitiva *MLMEX-DELETETXPROFILE.confirm* é gerada pela MLME de modo a indicar à WME o resultado do pedido efectuado com a recepção da primitiva *MLMEX-DELETETXPROFILE.request*. O campo *ResultCode* fornecido pela primitiva *MLMEX-DELETETXPROFILE.confirm*, indica se o perfil do transmissor foi apagado com sucesso (*SUCCESS*) ou se o canal indicado por *Channel Identifier* foi considerado inválido (*INVALID_PARAMETERS*). A figura 3.12 ilustra a interacção entre WME e a MLME quando a primitiva *MLMEX-DELETETXPROFILE.request* é gerada.

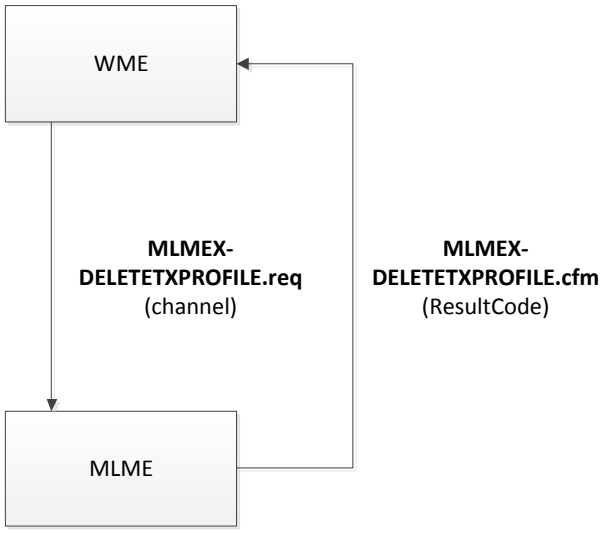


Figura 3.12: Requisição da eliminação do registo de um perfil de transmissor para um SCH

3.7.5 MLMEX-CANCELTX

A primitiva *MLMEX-CANCELTX.request* é gerada pela WME pedindo à MLME o cancelamento da transmissão na fila de transmissão correspondente ao campo *ACI* para um SCH indicado pelo campo *Channel Identifier*. O campo *ACI* é o índice da AC que corresponde a uma das filas de transmissão configuradas. Depois da recepção desta primitiva, a MLME deve descartar todos os pacotes no caminho de transmissão, cuja prioridade corresponda à AC indicada pelo campo *ACI* para o canal correspondente ao campo *Channel Identifier*.

A primitiva *MLMEX-CANCELTX.confirm* é gerada pela MLME de modo a indicar à WME o resultado do pedido efectuado com a recepção da primitiva *MLMEX-CANCELTX.request*. O campo *ResultCode* fornecido pela primitiva *MLMEX-DELETETXPROFILE.confirm*, indica se o perfil do transmissor foi apagado com sucesso (*SUCCESS*), se algum dos parâmetros passados foi considerado inválido (*INVALID_PARAMETERS*) ou se houve uma falha por outro motivo não especificado (*UNSPECIFIED_FAILURE*). A figura 3.13 ilustra a interacção entre WME e a MLME quando a primitiva *MLMEX-CANCELTX.request* é gerada.

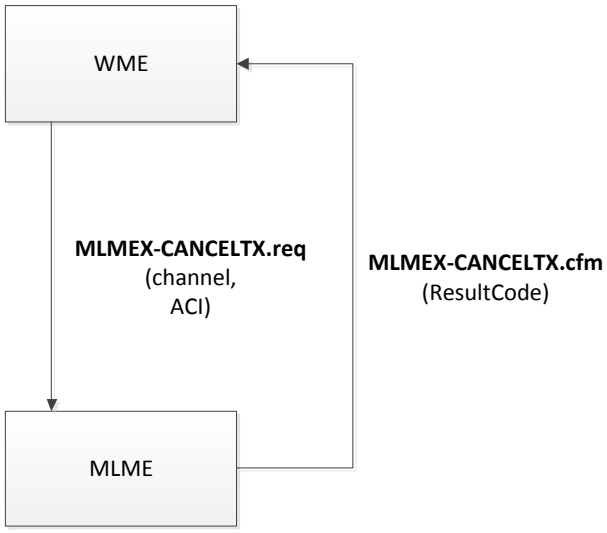


Figura 3.13: Requisição do cancelamento de uma fila de transmissão para um SCH

Capítulo 4

Software Base 802.11 em Linux

4.1 Introdução

Este projecto utiliza como base da sua implementação o software de livre acesso disponibilizado pelo grupo *Linux Wireless*. Este grupo desenvolve e mantém o código que implementa o sub-sistema wireless do Linux e fornece *drivers* necessários para a utilização de algumas placas comercializadas por certos fabricantes e suporte a algumas normas para o acesso *wireless*. No caso deste projecto, o *driver* utilizado é denominado por *ath5k* que oferece a interface necessária à utilização do *chipset* AR5414A-B2B, desenvolvido pela *Atheros*. O *driver ath5k* oferece suporte às placas desenvolvidas pela *Atheros* para versões do *chipset* AR5xxx, surgindo daqui o nome *ath5k*. Para além dos *drivers* disponibilizados para a interface entre as camadas superiores com o hardware, o software disponibilizado pela *Linux Wireless* apresenta as diferentes camadas de rede e *Application Programming Interface* (API) que oferecem a interface entre os vários blocos que constituem a arquitectura da sua implementação.

O *driver ath5k* apresenta vários modos de utilização do meio *wireless* [31]:

- Modo Ponto de acesso (*Access Point* (AP));
- Modo Estação (*Station* (STA));
- Modo *Ad-Hoc*;
- Modo *Mesh*.

Dos diferentes modos de utilização do meio *wireless* que o *driver* disponibiliza, o modo *ad-hoc* constitui aquele que interessa a este projecto, visto que um dispositivo WAVE estará a funcionar em modo *ad-hoc* alterado, como veremos mais à frente neste documento.

O *driver ath5k* suporta neste momento as normas IEEE 802.11 a/b/g e barramento PCI/*Peripheral Component Interconnect Express* (PCI-E)/*Personal Computer Memory Card International Association* (PCMCIA) [32]. Para a implementação deste projecto tiramos partido da implementação existente da norma IEEE 802.11, modificando-a para IEEE 802.11p.

4.2 Estrutura

A figura 4.1 apresenta os vários blocos que descrevem a divisão do *kernel* do *Linux* [33], desde o hardware até a aplicação. Os *WLAN device drivers* são divididos em dois módulos, módulo dependente do hardware e módulo referente ao protocolo. O módulo dependente do hardware difere consoante o fabricante e é baseado nas capacidades fornecidas pelo hardware. O módulo referente ao protocolo abrange a maior parte das funcionalidades que a sub-camada MAC oferece, respeitando a norma IEEE 802.11, este módulo é denominado por *mac80211* de implementação *softMAC*, visto que a implementação deste módulo apenas depende do software. Assim, os módulos superiores utilizam as funcões fornecidas pelo módulo *mac80211* para comunicar com o módulo dependente do hardware de modo a obter informação acerca do mesmo ou configurá-lo [33].

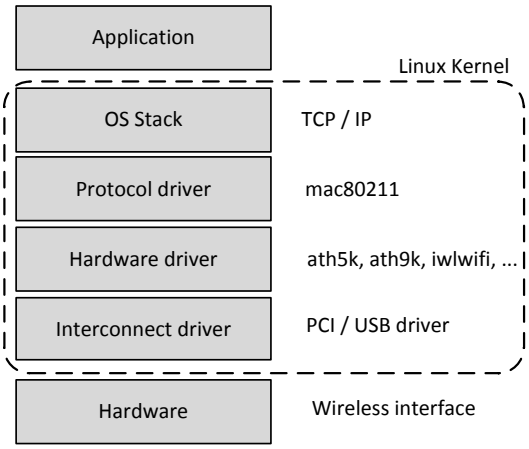


Figura 4.1: Linux Kernel Stack

Na figura 4.2 podemos ver a estrutura modular do software utilizado na implementação deste projecto. De seguida abordaremos estes diferentes módulos de modo a perceber, de

uma maneira geral, o que este software nos fornece.

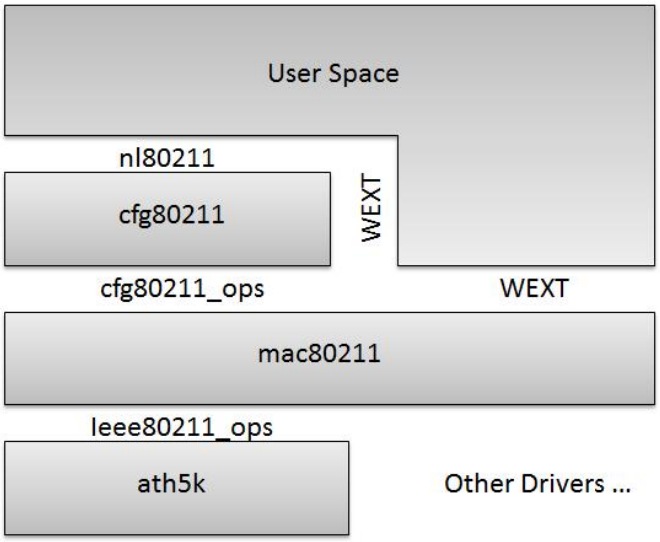


Figura 4.2: Arquitectura do software base utilizado

4.2.1 ath5k

O módulo *ath5k* corresponde ao *driver* dependente do hardware, para o caso deste projecto. Este *driver* faz a ponte entre o módulo *mac80211*, correspondente ao *softMAC*, e o hardware. Neste módulo são inicializadas as estruturas de controlo do hardware e as estruturas referentes às interfaces virtuais da camada física. As rotinas de tratamento às interrupções, como por exemplo a *tasklet* destinada ao envio de *beacons*, inicialização das filas de transmissão/recepção e inicialização/tratamento aos dispositivos que utilizam placas PCI, são definidas neste módulo.

Pode-se destacar a estrutura *ath5k_softc* que contém informação sobre o estado do dispositivo, sendo inicializado para cada interface virtual da camada física. Esta estrutura apresenta campos importantes à gestão do sistema, como o campo *pdev* que descreve o dispositivo PCI inicializado, correspondente à placa *wireless* identificada por um número atribuído pela rotina de inicialização da PCI. Através desta estrutura temos acesso às linhas de interrupção atribuídas, à informação acerca do canal que o hardware se encontra ou a uma estrutura geral destinada à informação e gestão do estado do hardware, denominada *ieee80211_hw*. Esta última estrutura, é utilizada para a passagem de informação nas API's, sendo definida no módulo *mac80211* para interacção entre este e o módulo *ath5k* através das *callbacks* definidas na estrutura *ieee80211_ops*. A estrutura *ieee80211_ops* contém várias *callbacks* que o *driver*

ath5k terá que tratar, como, por exemplo, a configuração do hardware para operar num novo canal ou para transmitir uma trama. Mais à frente no capítulo 5 referente à implementação, será abordado este módulo de forma mais aprofundada.

4.2.2 mac80211

O módulo *mac80211* trata da interacção do módulo *cfg80211* com os módulos de tratamento de *drivers* dependentes do hardware. No caso deste projecto, este módulo faz a ponte entre o módulo *cfg80211* com o módulo *ath5k*. *mac80211* apresenta estruturas e rotinas destinadas à gestão e solicitação de configuração do hardware. Neste módulo encontra-se implementado todo o conjunto de rotinas e estruturas para dispositivos destinados à comunicação sem fios respeitando a norma IEEE 802.11. O módulo *mac80211* depende do módulo *cfg80211* para o registo de um dispositivo na rede e para a configuração do mesmo. A MLME encontra-se implementada neste módulo, portanto todas as principais funcionalidades da sub-camada MAC são tratadas neste módulo, como autenticação, associação e *beaconing*. Este módulo apresenta ainda a implementação dos algoritmos de controlo da taxa de transmissão, *Minstrel*, *Minstrel-HT* e *Proportional Integral Derivative* (PID), e fornece funções para alocação/remoção de uma interface para um dispositivo e trata de grande parte do caminho de recepção/transmissão de dados.

O caminho de transmissão é ilustrado pela figura 4.3. Uma trama passada à sub-camada MAC é tratada pela função de tratamento da interface virtual *wireless* correspondente. De seguida, esta função converte a trama para o formato descrito na norma IEEE 802.11, identificando o tipo de trama, se corresponde a uma trama de dados ou de gestão, por exemplo. O próximo passo do caminho de transmissão corresponde à chamada das funções de tratamento da trama, que terá diferentes tratamentos consoante o seu tipo. Uma trama de dados, por exemplo, deverá passar por várias validações, visto que dependendo o modo *wireless* que a interface está a operar, esta pode requerer que o dispositivo esteja associado, por exemplo. Depois das diferentes validações a que a trama é submetida, esta é passada ao módulo *ath5k*, que cria toda a estrutura de controlo para a transmissão da mesma. Neste nível, as funções de tratamento da transmissão no módulo *ath5k* terão que configurar os descritores da transmissão, indicando a taxa de transferência e a potência de transmissão, por exemplo, que devem ser configuradas no hardware para o envio da trama.

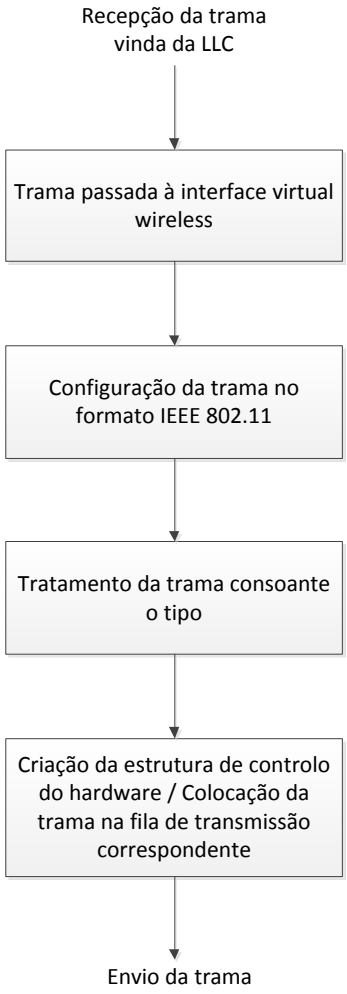


Figura 4.3: Caminho de transmissão do software base

O caminho de recepção é ilustrado pela figura 4.4. Quando uma trama é recebida pelo hardware, a função de tratamento à mesma é chamada pela rotina de interrupção. Depois disso, são reunidas informações acerca da trama e do destino da mesma, sendo tratada por diferentes funções, consoante o seu tipo. Toda a trama recebida é tratada consoante o modo *wireless* a que a interface virtual *wireless* destino está configurada. Uma trama de dados, por exemplo, deve ser sujeita a diferentes validações, como o estado da associação do dispositivo. Depois de passar por estas validações, a trama é configurada no formato referente à sub-camada LLC, como descrito pela norma IEEE 802.3. O último passou consiste na passagem da trama para a LLC.

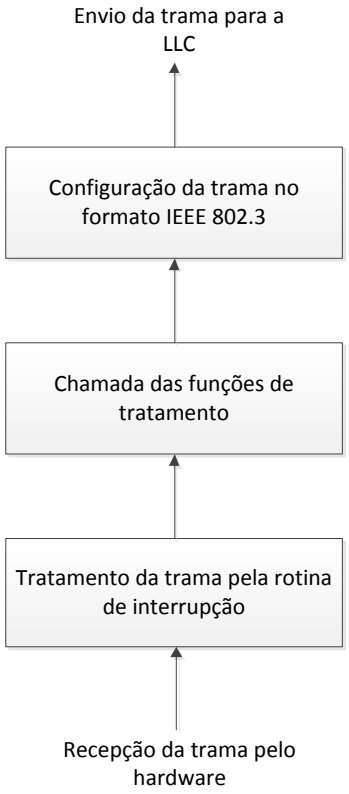


Figura 4.4: Caminho de recepção do software base

Os caminhos de recepção e transmissão serão abordados de forma mais aprofundada no capítulo 5. Através das *callbacks* definidas na estrutura *cfg80211_ops*, o módulo *cfg80211* interage com o módulo *mac80211* solicitando as suas funções. A estrutura *cfg80211_ops* apresenta várias *callbacks* destinadas à adição/remoção de uma estação ou mudança de canal, por exemplo, quando solicitado por *userspace*.

4.2.3 cfg80211 e nl80211

O módulo *cfg80211* consiste no conjunto de API's destinada à configuração, substituindo assim o módulo anteriormente utilizado, *Wireless-Extensions* (WEXT). O bloco WEXT continua a existir para suportar antigas implementações que o utilizam, mas não haverá mudanças no código relacionado ao mesmo. Assim, o módulo *cfg80211* constitui o presente e o futuro para qualquer desenvolvimento de software para o meio *wireless* [34].

O módulo *cfg80211*, juntamente com o módulo *mac80211*, providencia funções para con-

figuração e controlo dos dispositivos *wireless*. Este módulo apresenta um conjunto de API's que providenciam funções para o registo de um dispositivo no sistema, gestão de uma STA ou *Mesh*, por exemplo, através da comunicação entre o *userspace* e a sub-camada MAC propriamente dita. Este módulo faz a validação dos parâmetros que devem ser passados à sub-camada MAC para efectuar devidamente a função requerida, como por exemplo o registo de um dispositivo requer informação acerca da banda, canal, etc., que este suporta. *cfg80211* faz o controlo das regras referentes ao domínio que nos encontramos, visto que de país para país, as regras relativas à banda de frequência utilizada variam.

Para configurar um dispositivo, o módulo *cfg80211* utiliza as API's *nl80211*. *nl80211* é um conjunto de API's que utilizam a interface de comunicação *kernelspace-userspace netlink*. *Netlink* consiste numa interface padrão baseada em *sockets* para processos em *userspace* e uma API interna do *kernel* para módulos do *kernel* [35]. *Netlink* é um dos vários métodos *Inter-Process Comunication* (IPC) que existem utilizados para transferência de informação entre o *kernelspace* e o *userspace*, assim como as chamadas ao sistema (*syscalls*), *ioctl* ou sistema de ficheiros *proc* [36].

Netlink foi o método IPC adoptado devido à facilidade na utilização do mesmo, sendo necessário apenas a adição de uma contante, o tipo de protocolo, ao ficheiro *netlink.h* presente no *kernel*. A partir daí, o módulo *kernel* e a aplicação podem comunicar utilizando *sockets* imediatamente, enquanto que a adição de novas tarefas para chamadas ao sistema, *ioctl* ou ficheiros *proc*, não é trivial [36].

Outra vantagem do *netlink* consiste no facto deste ser assíncrono, visto que, como todas as *socket API*, esta disponibiliza uma fila para o armazenamento de mensagens, com o objectivo de suavizar a recepção de mensagens. *Netlink* suporta *multicast* que consiste noutra vantagem sobre as chamadas ao sistema, *ioctl* e ficheiros *proc*. Um processo pode enviar uma mensagem em *multicast* para um endereço de um grupo ("família") *netlink*, e um número qualquer de outros processos podem auscultar este endereço [36]. Assim, a partir de uma mensagem conseguimos chegar a vários processos, sendo esta processada de várias maneiras diferentes.

Netlink apresenta ainda a vantagem de ser um método IPC bidireccinal, ou seja, permite a inicialização de uma secção por aplicações em *userspace* e por algum módulo do *kernel*. Isto não acontece com as chamadas ao sistema, por exemplo, que só permitem a inicialização de uma sessão apenas por aplicações em *userspace* [36].

Apesar de todas as vantagens na utilização do *netlink*, existem contrapartidas a considerar [37]:

- Cada entidade que utilize *sockets netlink* tem que definir o seu próprio tipo de protocolo, ou família, no ficheiro *netlink.h*, sendo necessária a recompilação do *kernel* antes de poder ser utilizado;
- O número máximo de famílias *netlink* está fixado em 32.

Com vista à eliminação destas contrapartidas foi implementada a família *Generic Netlink*, onde a sua arquitectura é ilustrada pela figura 4.5. As comunicações *generic netlink* são essencialmente uma série de diferentes canais de comunicação que são multiplexados numa única família *netlink*. Estes canais de comunicação são identificados por um número que são dinamicamente alocados pelo controlador *generic netlink*. Este controlador é inserido como um utilizador especial do *generic netlink* e ausculta um canal de comunicação fixo, número 0x10, previamente alocado na inicialização do mesmo, estando sempre presente no *kernel* [38]. Assim, um módulo do *kernel* ou uma aplicação que oferece serviços através do barramento *generic netlink*, estabelecem novos canais de comunicação registando os seus serviços através do controlador *generic netlink*. Os utilizadores que queiram usufruir de um serviço, consultam o controlador para saber se o serviço existe e qual o número correcto do canal onde este serviço é disponibilizado [37].

Cada família *generic netlink* oferece diferentes atributos e comandos para a interação com um módulo do *kernel* ou com uma aplicação. Estes comandos são anexados a *callbacks* no módulo do *kernel* e devem receber mensagens com diferentes atributos associados. Estes atributos são identificadores que guardam uma informação específica, sendo registados de forma a, a partir destes identificadores, tornar mais fácil o acesso à informação contida na mensagem *netlink* [37]. Veremos mais à frente, no capítulo 5, de forma mais aprofundada, como podemos manipular e registar comandos e atributos de uma mensagem *netlink*.

O *nl80211* implementa o mesmo conceito descrito anteriormente, sendo que aloca a família *generic netlink* com nome *"nl80211"* e anexa os comandos identificados em *nl80211_commands* no ficheiro *nl80211.h* presente no *include* do software utilizado na implementação deste projecto. A utilização destes comandos será discutida no capítulo 5 desta dissertação.

Um exemplo de implementação de aplicações que utilizam o *nl80211* é *iw*. *iw* é uma *Command-Line Interface* (CLI) baseada no *nl80211* utilizada para configuração de dispositivos *wireless*. Esta aplicação está ainda em desenvolvimento, mas já apresenta muitas funcionalidades e alternativas ao *iwconfig*, aplicação que utiliza o WEXT [39].

Neste trabalho foram utilizados comandos existentes da aplicação *iw* e inseridos novos comandos, de forma a ser possível comunicar, a partir do *userspace*, com a sub-camada MAC.

Os novos comandos inseridos têm o objectivo de oferecer ao utilizador ferramentas para utilizar as funcionalidades especificadas na norma IEEE 1609.4, implementadas no trabalho descrito nesta dissertação. Mais à frente, na secção 5.7 do capítulo 5 desta dissertação, serão descritas as alterações efectuadas à aplicação *iw* na implementação deste projecto.

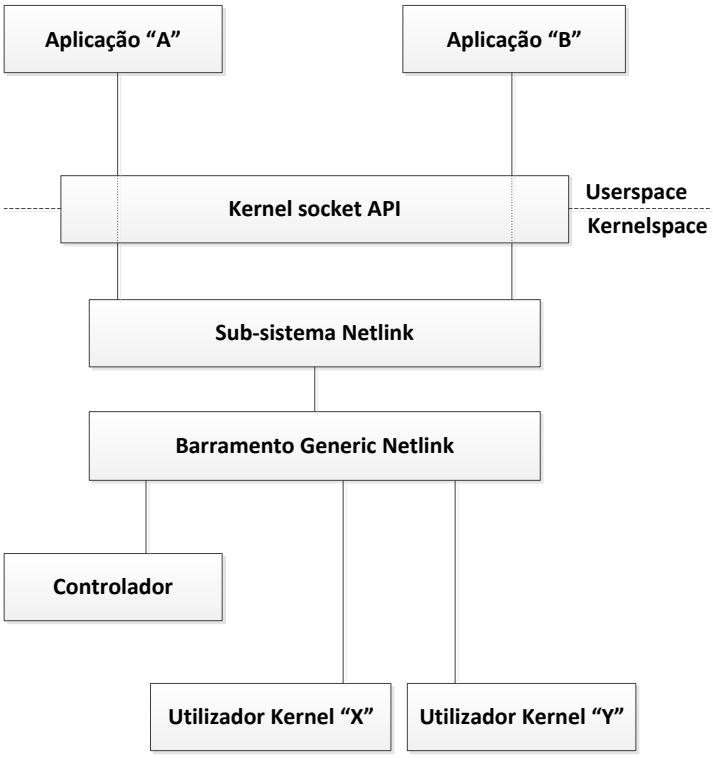


Figura 4.5: Arquitectura do *generic netlink* como barramento de comunicação [38]

Capítulo 5

Implementação em Linux das Normas IEEE 802.11p/1609.4

5.1 Introdução

A implementação da sub-camada MAC WAVE consiste na alteração do software base disponibilizado pela *Linux Wireless*, como vimos anteriormente, de modo a atingir as exigências principais descritas nas normas IEEE 802.11p e IEEE 1609.4. A fase inicial desta implementação consistiu na leitura do código disponibilizado pela *Linux Wireless* e compreensão de como está organizada a *stack wireless* e onde realizar as alterações pretendidas de modo a atingir o objectivo proposto para este projecto. Esta fase de aprendizagem foi efectuada através de *tracing* do código, na grande maioria por colocação de *printk*'s em pontos específicos do código, e recurso a ferramentas de pesquisa, como por exemplo o *cscope*. Depois de verificado o estado da implementação original do *driver*, dividiu-se a implementação das alterações ao *driver* por objectivos:

- **Modo *wave***
  - Criação de um novo modo *ad-hoc* alterado, a que se chamou *wave*, oferecendo ao utilizador um modo *wireless* com todas as funcionalidades implementadas neste trabalho;
- **Qualidade de Serviço**
  - Alteração dos parâmetros EDCA como indicado na norma IEEE 802.11p;

- **Caracterização do CCH e SCH**
  - Descartar pacotes IP quando a interface virtual está configurada no CCH (178);
  - Criação de um campo relativo aos perfis de transmissão para cada SCH e verificação da existência do mesmo quando se tenta transmitir dados IP num SCH.
- **Comutação de canais**
  - Duplicação da interface física relativa à placa *AR5414* com o intuito de haver uma interface configurada no CCH e outra num SCH;
  - Implementação das funções necessárias à comutação de canal entre a interface configurada no CCH e a interface configurada no SCH.
- **Sincronização**
  - Implementação de um módulo para interface com um receptor GPS;
  - Implementação de um filtro de Kalman.
- **Primitivas e *iw***
  - Implementação das primitivas necessárias ao correcto funcionamento da norma IEEE 1609.4;
  - Implementação de novos comandos para a aplicação *iw* de modo a permitir a configuração/verificação da sub-camada MAC em *userspace*.

A arquitectura desta implementação é ilustrada na figura 5.1. Esta figura mostra os três módulos, *ath5k*, *mac80211* e *cfg80211*, e como estes interagem. Como se pode observar, cada módulo tem uma extensão que representam as alterações efectuadas de modo a suportar a sub-camada MAC WAVE, juntamente com a sub-camada MAC já implementada. Uma das funcionalidades mais importantes já existente no módulo *mac80211* é o encaminhamento do pacote para a fila relativa à prioridade que acompanha o pacote, ilustrado também na figura 5.1. Assim, na extensão do módulo *mac80211* foi apenas necessário alterar os parâmetros EDCA para os valores exigidos pela norma IEEE 802.11p. A coordenação de canais apresenta-se como a funcionalidade mais crítica na implementação da extensão da sub-camada MAC para dispositivos WAVE. Esta funcionalidade também encontra-se ilustrada pela figura 5.1, como parte da extensão do módulo *ath5k*, sendo abordada mais à frente na secção 5.5. O objectivo desta figura é ilustrar toda a interacção entre os vários módulos implementados e

ainda as camadas superiores e o hardware. Note-se que a sub-camada MAC WAVE apresenta-se como o conjunto da implementação existente anteriormente no software disponibilizado pela *Linux Wireless* e as alterações feitas no trabalho descrito por esta dissertação.

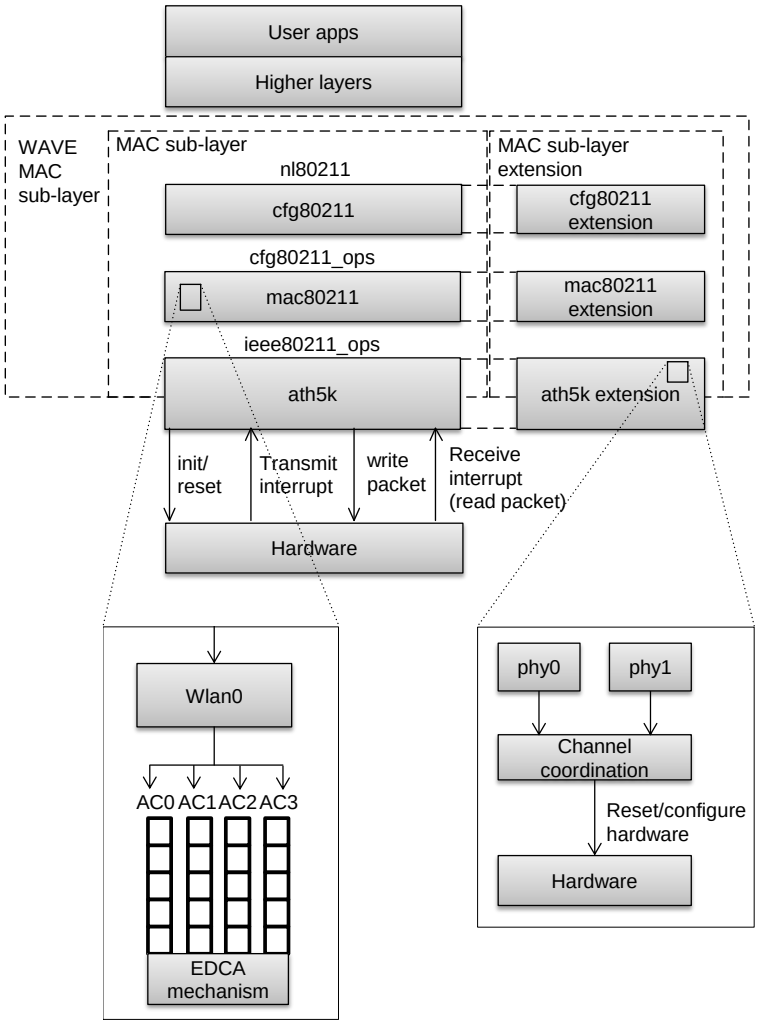


Figura 5.1: Arquitectura da implementação da sub-camada MAC WAVE

5.2 Modo *wave*

O *driver ath5k* disponibiliza o modo *ibss* (*ad-hoc*) implementado. Visto que um dispositivo WAVE deve conseguir comunicar sem estar associado, foi criado um novo modo denominado *wave*, com base no modo *ibss* (*ad-hoc*) presente no *driver*. Este novo modo *wireless* consiste numa cópia do modo *ad-hoc* normal, mas com algumas alterações no caminho de transmissão

e no caminho de recepção. Estas alterações foram efectuadas de modo a conferir a capacidade de transmissão/recepção sem necessidade de qualquer associação por parte do dispositivo.

O pacote a ser transmitido é passado à sub-camada MAC pelas camadas superiores através da estrutura *sk_buff* (*socket buffer*), que consiste na estrutura fundamental no tratamento da rede do *Linux*. Utilizada no caminho de recepção e transmissão de pacotes, esta estrutura representa uma fila onde se encontram armazenadas as informações acerca do protocolo de rede a que o pacote baseia-se, assim como o conteúdo do pacote. O caminho de transmissão é ilustrado pela figura 5.2, que mostra as funções invocadas mais relevantes no caminho de transmissão e a estrutura *ieee80211_tx_data* (ficheiro *ieee80211_i.h*). No caminho de transmissão é recolhida informação acerca da interface virtual, o canal, a taxa de transmissão e *flags* de controlo, entre outros parâmetros, em que será transmitido o pacote. Esta recolha de informação é efectuada pela função *ieee80211_tx_prepare* (ficheiro *tx.c*) e a informação, assim como o ponteiro para a posição do *socket buffer* referente ao pacote, é guardada na estrutura *ieee80211_tx_data*, dado pelo campo *skb* como podemos ver na figura 5.2.

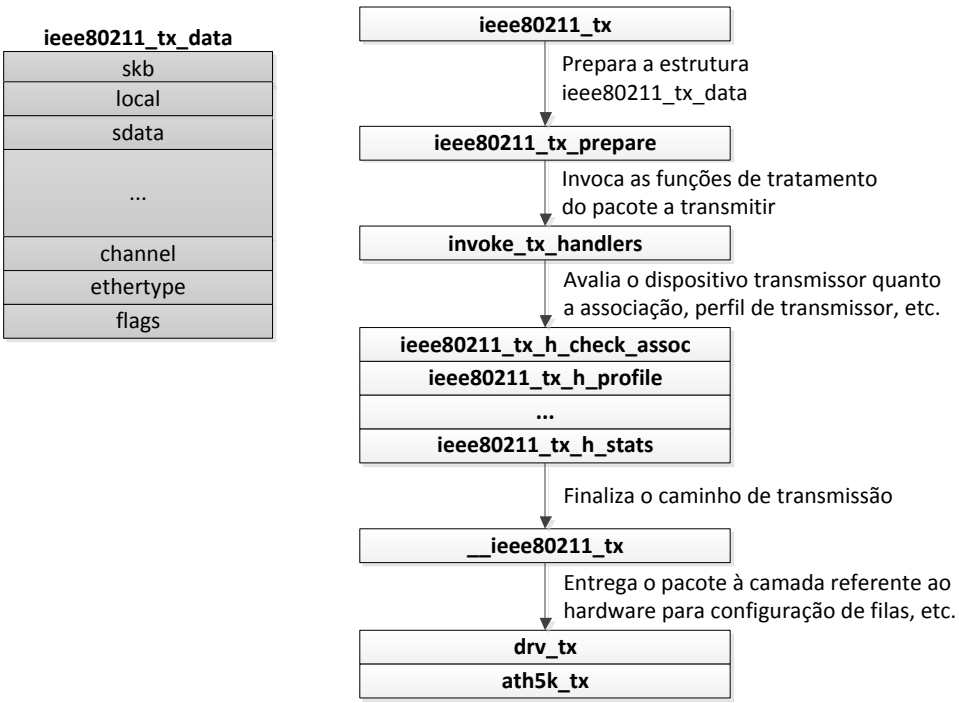


Figura 5.2: Caminho de transmissão

A estrutura *ieee80211_sub_if_data* é uma das estruturas de dados mais importantes na implementação dos módulos que constituem a *stack wireless* do *Linux*. Esta estrutura contém toda a informação acerca da interface virtual *wireless*, hardware e estado do sistema, como se pode observar na figura 5.3. A figura 5.3 ilustra as estruturas mais utilizadas e a interacção entre as mesmas, através das quais podemos obter informação acerca de todo o sistema e comunicar entre os vários módulos a partir das API's implementadas.

Depois de reunida a informação necessária, o *driver* chama a função *invoke_tx_handlers* (ficheiro *tx.c*) (figura 5.2), que invoca funções destinadas à verificação do estado do dispositivo, sendo que a função *ieee80211_tx_h_check_assoc* (ficheiro *tx.c*) é destinada à verificação do estado de associação do dispositivo. A partir da estrutura *ieee80211_tx_data* (ficheiro *ieee80211_i.h*) temos acesso ao campo *struct ieee80211_sub_if_data *sdata* (ficheiro *ieee80211_i.h*), como podemos observar na figura 5.2. A partir do campo *sdata* facilmente consegue-se verificar se se trata de um dispositivo configurado no modo *wave* a partir do campo *vif*. Por sua vez, a estrutura *ieee80211_vif* (ficheiro *mac80211.h*) contém o campo *type* que indica o modo a que a interface encontra-se configurada. Este encadeamento de campos dentro das várias estruturas é ilustrado pela figura 5.3. Assim, para não ser necessária associação basta haver permissão para transmissão e recepção de dados sem verificação, ou seja, para dispositivos configurados no modo *wave* a função *ieee80211_tx_h_check_assoc* não é chamada. Um dispositivo está configurado no modo *wave* se o campo *type* da estrutura *ieee80211_vif* for igual a *NL80211_IFTYPE_WAVE*, tipo de interface introduzido no ficheiro *nl80211.h*.

Além da preocupação em relação à associação, a criação do modo *wave* implicou ainda a criação de uma nova banda de frequências aquando da inicialização da estrutura que contém informação acerca do hardware, *ieee80211_hw* (ficheiro *mac80211.h*). A figura 5.4 mostra os principais campos da estrutura *ath5k_softc* e o caminho de inicialização de uma interface física perante o driver *ath5k* para o caso deste projecto, visto que utilizamos uma placa mini-PCI. A função *ath5k_setup_bands* (ficheiro *base.c*) é chamada pela função de inicialização do módulo *ath5k* e insere as diferentes bandas suportadas pelo *driver*, existindo dois tipos: *IEEE80211_BAND_2GHZ* e *IEEE80211_BAND_5GHZ*. Foi introduzida em *enum ieee80211_band*, no ficheiro *cfg80211.h*, uma nova banda a que chamou-se *IEEE80211_BAND_5_9GHZ*. A cada banda são correspondidos os canais respectivos através da chamada à função *ath5k_setup_channels*. É nesta função que anexamos os canais correspondentes à banda de frequência para DSRC a 5.9 GHz. Na função *ath5k_setup_bands* anexam-se também os *bitrates*

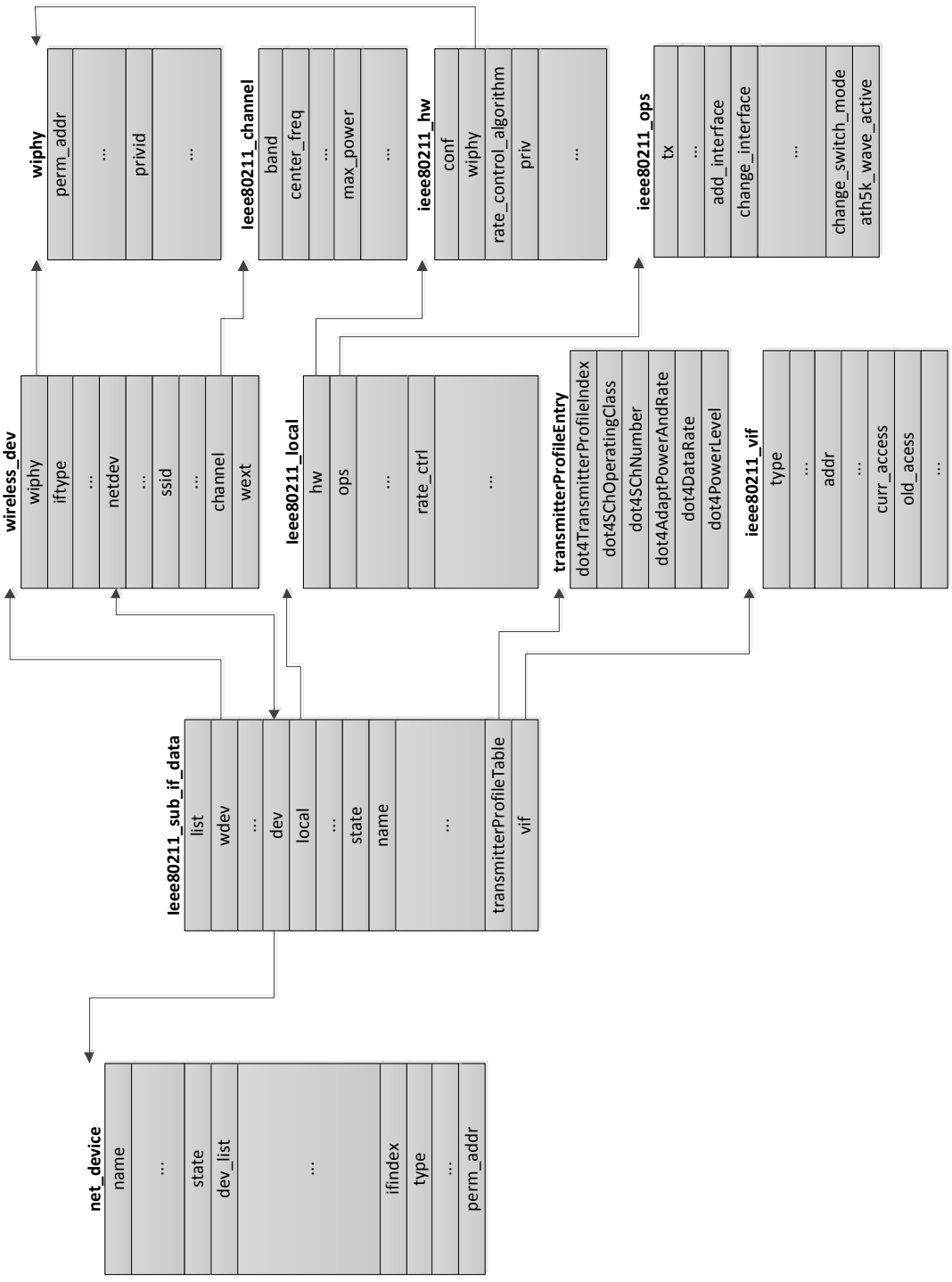


Figura 5.3: Estruturas mais relevantes que compõem a *stack* wireless

suportados por cada banda à estrutura referente ao hardware. Foram mantidos os valores correspondentes à norma IEEE 802.11a, mas que serão reduzidos à aproximadamente metade com a largura de banda do canal configurada a 10 MHz.

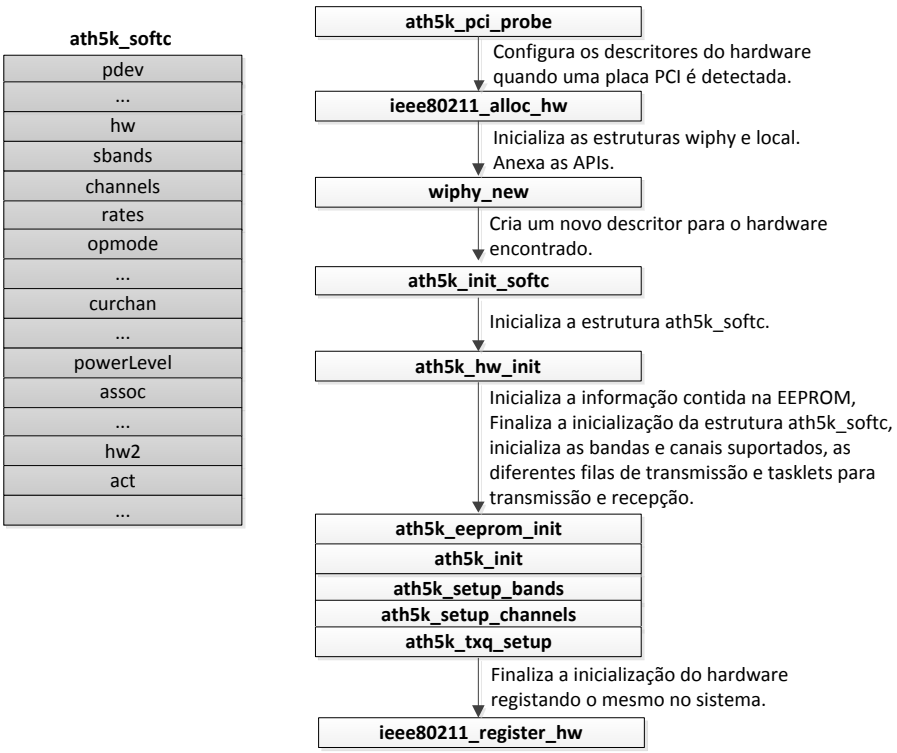


Figura 5.4: Estrutura *ath5k_softc* e caminho de inicialização do *driver ath5k*

Também foi necessário alterar a largura de banda visto que o valor da largura de banda configurada inicialmente é 20 MHz e, apesar de ser descrito na norma IEEE 802.11p que, em certos casos, os dispositivos poderão operar a uma largura de banda de 20 MHz, o valor padrão corresponde a 10 MHz. Quando a função de inicialização do hardware, *ath5k_hw_init* (ficheiro *attach.c*), é chamada, a largura de banda, assim como parâmetros como o canal inicial e modo em que a antena irá ser configurada, é configurada para *AR5K_BWMODE_DEFAULT*, que corresponde aos 20 MHz. Utilizando o modo *AR5K_BWMODE_10MHZ* definida em *enum ath5k_bw_mode* no ficheiro *ath5k.h*, configuramos o dispositivo com uma largura de banda de 10 MHz.

5.3 Qualidade de Serviço

As normas IEEE 802.11p e IEEE 1609.4 especificam que os parâmetros para utilização do mecanismo EDCA devem ser alterados segundo a tabela presente na norma IEEE 802.11p, como já foi referido anteriormente neste documento. Na função *ieee80211_add_iface* (ficheiro *cfg.c*) chamada quando uma nova interface é adicionada, foi adicionado uma condição para que se a interface for do tipo *wave* é corrido um ciclo 'for' de modo a configurar as quatro filas de transmissão com os valores presentes na tabela 3.2, recorrendo à função já existente *ieee80211_set_txq_params* (ficheiro *cfg.c*). Assim, sempre que for adicionada uma interface *wave*, os parâmetros EDCA padrão são configurados de acordo com a norma.

5.4 Caracterização do CCH e SCH

A norma IEEE 1609.4 especifica que, para um dispositivo WAVE ser capaz de transmitir pacotes IP num SCH, é necessário que um perfil de transmissor seja registado primeiro. No CCH não podem ser transmitidos pacotes IP. Para ser possível identificar um pacote IP é necessário verificar o campo *ethertype* associado à trama. Esta informação é extraída do campo *data* da estrutura *sk_buff*, campo que contém a trama guardada no *socket buffer*, e esta é copiada para o campo *ethertype* que faz parte da estrutura *ieee80211_tx_data* (figura 5.2). Assim, basta avaliar se o campo *ethertype* corresponde ao valor para pacotes IP indicado pelo identificador do protocolo *Ethernet*. Podemos ver na figura 5.5 que se a trama de dados corresponder a dados IP então o pacote é descartado, senão o processo continua normalmente.

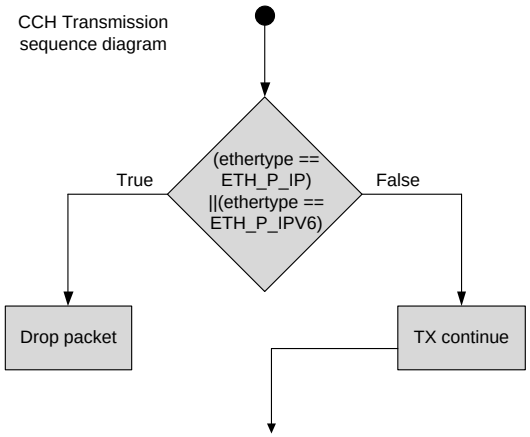


Figura 5.5: Caminho de transmissão no CCH

A figura 5.6 ilustra a situação de tentativa de transmissão de dados IP num SCH. Nesta situação, o número do SCH em questão é procurado na tabela de perfis de transmissor de modo a validar o processo. A implementação desta tabela consiste na definição da estrutura *transmitterProfileEntry* e na introdução do campo *transmitterProfileTable* na estrutura *ieee80211_sub_if_data*, referida anteriormente.

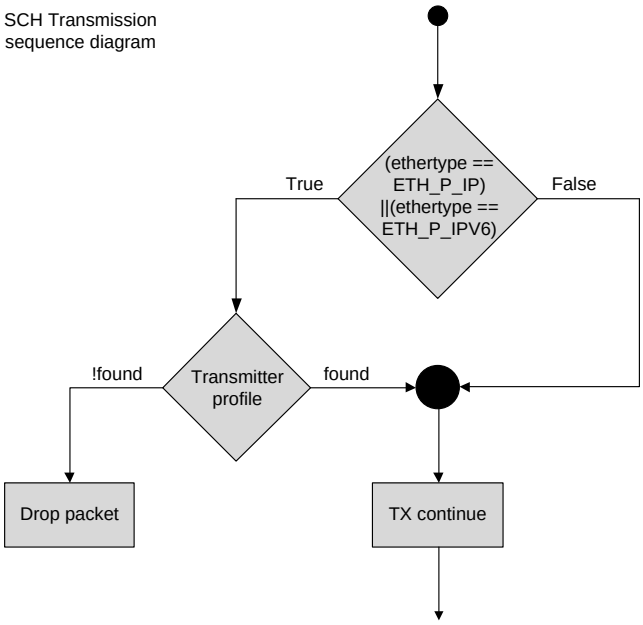


Figura 5.6: Caminho de transmissão no SCH

A estrutura *transmitterProfileEntry* foi criada com base na MIB presente na norma IEEE 1609.4, e corresponde a uma entrada da tabela de perfis do transmissor. Esta estrutura apresenta os seguintes campos:

- **dot4TransmitterProfileIndex**: número inteiro que corresponde ao índice do perfil na tabela;
- **dot4ServiceChannelNumber**: número inteiro que corresponde ao número do SCH;
- **dot4AdaptablePowerAndRate**: variável booleana que indica se os valores para a potência de transmissão e *bitrate* são adaptáveis ou não;
- **dot4DataRate**: número inteiro que corresponde ao *bitrate* que será configurado;

- **dot4PowerLevel**: número inteiro que corresponde ao nível da potência de transmissão que será configurada.

Como foi referido anteriormente, a estrutura *ieee80211_sub_if_data* corresponde a uma estrutura muito utilizada na implementação do sub-sistema *wireless* do *Linux*. Visto que necessitamos de introduzir a tabela de perfis do transmissor em uma estrutura que nos dê mobilidade ao longo do código, sendo de fácil acesso, escolhemos a estrutura *ieee80211_sub_if_data*, introduzindo um *array* com oito entradas, uma para cada canal da banda de frequências reservada para DSRC a 5.9 GHz, com o nome *transmitterProfileTable*, como se pode observar na figura 5.3.

A tentativa de transmissão de dados IP num SCH, implica a pesquisa na tabela de perfis, sendo o pacote descartado se nenhuma das entradas da tabela corresponder ao SCH em questão. Esta verificação é feita pela função *ieee80211_tx_h_profile* inserida como uma das funções de tratamento do pacote chamada pela função *invoke_tx_handlers* no caminho de transmissão (figura 5.2).

5.5 Comutação de canais

A norma IEEE 1609.4 especifica a função de coordenação de canais que é responsável pelo controlo dos diferentes intervalos temporais como vimos na figura 3.4 do capítulo 3, secção 3.3. A implementação desta função poderia seguir duas hipóteses: a duplicação da interface virtual wireless, normalmente registada com um nome semelhante a *wlan0*, ou a duplicação da interface física, normalmente registada com um nome semelhante a *phy0*. A primeira hipótese não se apresenta muito favorável, visto que no momento da troca de canal, toda a interface física correspondente ao hardware seria sujeita a um reset completo e reconfiguração, o que levaria algum tempo a efectuar-se. A segunda hipótese foi escolhida devido ao facto de termos não só duas interfaces físicas para o mesmo hardware, como também temos duas interfaces virtuais wireless anexadas a cada uma das interfaces físicas. Os módulos acima do *ath5k* tratam, portanto, as duas interfaces como se fossem completamente independentes. Assim, quando o sistema é carregado, a interface correspondente ao CCH é configurada no hardware e a interface correspondente ao SCH espera pela configuração pelo utilizador.

Depois de definida a abordagem a seguir, foi necessário duplicar a interface física relativa ao hardware *wireless* utilizado neste projecto. Desta forma, obtemos duas interfaces para o mesmo hardware configuradas em diferentes canais, sendo tratadas de forma independente.

Assim, fixamos uma interface no CCH e outra num SCH configurado pelo utilizador.

A interface física implementada no *driver ath5k* é uma estrutura chamada *ath5k_softc* (*Software Carrier*) definida no ficheiro *base.h* (figura 5.4). Esta estrutura mantém informação acerca do estado do *driver* e do hardware, apresentando-se como a estrutura mais importante e utilizada no módulo *ath5k*.

A estrutura *ath5k_softc* é criada pela função *ath5k_pci_probe* (ficheiro *pci.c*) chamada quando uma placa mini-PCI é detectada, como podemos observar na figura 5.4. No ficheiro *pci.c* do módulo *ath5k*, é definida uma tabela de identificadores de PCI's conhecidos, chamada *ath5k_pci_id_table*. Através de um processo de *debug*, concluimos que a placa *AR5414* apresenta o identificador 0x001b que corresponde à placa compatível *AR5413 Eagle*. A partir deste identificador conseguimos duplicar o nosso hardware *wireless*, registando o mesmo como um hardware "virtual". Para tal foi necessário criar um *Platform Device* de modo a registar o mesmo no sistema, visto que o nosso dispositivo *wireless* é detectado pelo barramento PCI.

Um dispositivo PCI colocado no barramento PCI pode dizer ao sistema que tipo de dispositivo se trata e onde estão localizados os seus recursos. Assim, quando o *kernel* é inicializado, o dispositivo apresenta informação através da estrutura *pci_device_id*, que fornece, entre outras informações, o nome do fabricante e identificador do dispositivo. Se o dispositivo for conhecido pelo *driver*, são criadas estruturas suporte para o funcionamento do mesmo. Este cenário é o que acontece com o dispositivo *wireless* utilizado neste projecto, mas como queremos criar um dispositivo "virtual", teremos que criar e registar um novo dispositivo de modo a ser tratado pelo *driver* como se de um dispositivo real se tratasse, sem haver conflitos. Como este registo será feito já na função de *probe* da placa mini-PCI, não é necessário a criação de funções para o preenchimento da estrutura *platform_driver*. Esta estrutura apenas teria que ser criada se quisessemos criar e registar um novo *Platform Driver* e não utilizar o já existente.

Para o registo deste utilizou-se a API existente no *kernel* chamada *platform_device_register*. Esta função tem como entrada um ponteiro para uma estrutura *platform_device* criada previamente com o nome *my_wifi_pdev*. Além da criação e registo deste dispositivo, foi necessário criar e alocar uma nova estrutura *ieee80211_hw* para manter o estado deste novo dispositivo, efectuado pelas funções *ieee80211_alloc_hw* e *ieee80211_register_hw*. Todo o caminho de inicialização é efectuado como ilustrado na figura 5.4.

De modo a manter a implementação flexível, foi adicionado um novo campo à estrutura *ath5k_softc*, chamado "hw2", sendo este referente ao dispositivo de hardware semelhante. Como cada interface física é representada por uma estrutura *ath5k_softc*, teremos para cada

estrutura *ath5k_softc* duas estruturas que descrevem o hardware, "hw" e "hw2". Assim, temos acesso à descrição do hardware para a interface configurada no CCH, por "hw" por exemplo, e à descrição do hardware para a interface configurada no SCH, por "hw2", na mesma estrutura *ath5k_softc*.

Com esta implementação a função *ath5k_init_softc* é chamada duas vezes para o mesmo dispositivo e serão registadas duas interfaces físicas, por exemplo "phy0" e "phy1" da figura 5.1, visíveis ao utilizador. Assim, o utilizador configurará a interface destinada ao SCH e a função responsável pela coordenação de canais configurará o hardware para a interface referente ao intervalo de canal correspondente. Com esta abordagem acabamos por não ter propriamente uma comutação de canais, mas sim uma comutação de interfaces, que permanecem configuradas da mesma maneira até nova ordem do utilizador.

De modo a efectuar a comutação de canais foi criado um novo campo na estrutura *ath5k_softc* chamado "act". Este campo refere-se à estrutura *ath5k_softc* que está configurada no hardware no momento. A utilização deste método facilita a recolha de informação acerca da configuração actual do dispositivo. Na presença de uma tentativa de transmissão, a interface que encontra-se inactiva colocará em fila os pacotes a serem transmitidos, ficando em espera até ao intervalo do canal a que lhe diz respeito.

A figura 5.8 ilustra a máquina de estados que implementa a função de coordenação de canais que consiste na criação de dois *timers*, um destinado ao controlo do intervalo de canal e outro para o controlo do intervalo de guarda. O *Timer1* é responsável pelo controlo do intervalo de canal, sendo activado a cada 50 ms. O *Timer2* controla o intervalo de guarda e é activado pelo *Timer1* no início de cada intervalo de canal.

No início de um intervalo de canal, o intervalo de guarda começa e, como podemos observar na figura 5.7, a transmissão deve ser desactivada para a interface activa. Quando o *Timer1* é activado, a função referente ao mesmo, à qual chamamos *ath5k_reset_switch_channel_timer*, pára as filas de transmissão do hardware através da chamada da função *ieee80211_stop_queues*, já existente no *driver*. O estado inicial corresponde a *GI_SYNC_1* (*Guard_Interval_Synchronization_1*), que indica o primeiro intervalo de sincronização. O *Timer2* é rearmado para 1 ms e o estado passa a ser *GI_MAX_SWITCH* (*Guard_Interval_Max_Switch*) correspondendo ao intervalo de comutação de canais (*MaxChSwitchTime*). Quando a função *ath5k_reset_switch_channel_timer* é chamada novamente, é requerido a reinicialização do hardware. Uma vez que para dispositivos que utilizam o modo WAVE implementado neste projecto é necessário tratar o conceito introduzido sobre a interface activa, não podemos utilizar a função de reinicial-

ização do hardware, *ath5k_reset*, já existente no *driver*. Implementou-se então a função *ath5k_reset_switch_channel* semelhante à função *ath5k_reset*, mas com controlo da interface activa. Esta função de reinicialização liberta as filas de transmissão, reinicia o hardware e inicializa as filas de recepção para a interface correspondente ao novo canal, que passará a estar activa. No final deste estado o *Timer2* é novamente rearmado para 2 ms e o estado passa a ser *GI_SYNC_2* (*Guard_Interval_Synchronization_2*). Neste estado são activadas as filas de transmissão e o intervalo de guarda termina. Passado 1 ms o *Timer2* é novamente activado e o estado passa a ser o inicial, *GI_SYNC_1*. A implementação do *Timer1* referido anteriormente será abordada de seguida na secção 5.6.

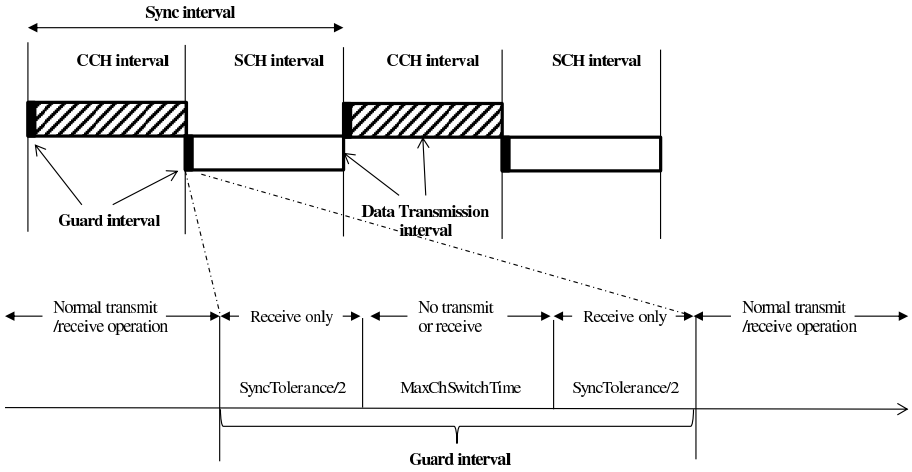


Figura 5.7: Diferentes intervalos de tempo (acesso alternado) e decomposição do intervalo de guarda

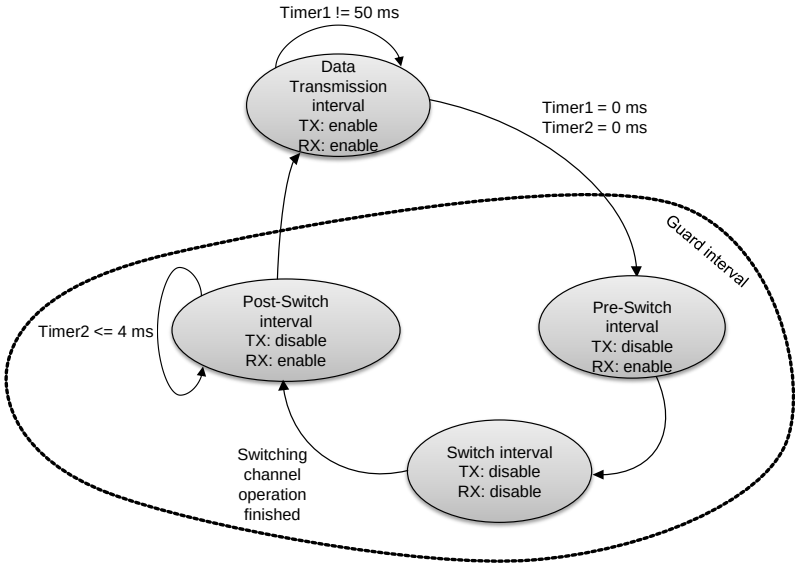


Figura 5.8: Máquina de estados referente à implementação da função de coordenação de canais

5.6 Sincronização

5.6.1 O módulo GPS e a interacção com o módulo *ath5k*

Como foi descrito no capítulo 3, secção 3.5, a sub-camada MAC possui uma função de sincronização que utiliza o UTC obtido através de um dispositivo receptor GPS. A interface entre o módulo GPS e o *Linux* é feita a partir de um *driver* que detecta e trata o sinal recebido pelo receptor GPS. O *driver* utilizado que efectua esta interface denomina-se *LinuxPPS*.

Este *driver* suporta a leitura do sinal PPS, assim como do tempo GPS. O sinal PPS corresponde ao sinal eléctrico enviado pelo receptor GPS a cada segundo, indicando com uma precisão elevada o início de um segundo. O tempo GPS corresponde à hora actual do dia, como por exemplo 12:34:56 UTC, sendo apresentado na forma *National Marine Electronics Association* (NMEA) [40].

A figura 5.9 ilustra de forma genérica como podemos obter o sinal PPS a partir de um receptor GPS. A partir do *polling* a um pino *General Purpose Input/Output* (GPIO) da placa de desenvolvimento, conseguimos obter o sinal PPS que é tratado por uma função do módulo criado pelo *driver LinuxPPS* para o tratamento de informação vinda de um pino GPIO. A função que efectua o *polling* é controlada por um *timer* activado de 1 em 1 ms, verificando se

um sinal PPS é recebido no pino GPIO. Quando este detecta um sinal PPS, o *timer* efectua uma espera de 950 ms até ser novamente activado, visto que não será enviado nenhum sinal PPS durante aproximadamente 1 s.

O sinal PPS também pode ser obtido a partir do pino *Data Carrier Detect* (DCD) *Recommended Standard-232* (RS-232), se existente no receptor GPS utilizado, sendo que esta gera uma interrupção que activa a função destinada ao seu tratamento no *driver*. Independentemente da abordagem escolhida para a obtenção do sinal PPS, a implementação da interface entre o *driver LinuxPPS* e o *driver ath5k* é semelhante.

A interacção entre o *driver LinuxPPS* e o *driver ath5k* descrita nesta secção, é ilustrada pela figura 5.10. Quando um sinal PPS é detectado no pino escolhido para o efeito, a função de tratamento do mesmo é chamada. Assim, foi implementado um novo *timer*, denominado *output_timer* que corresponde ao *Timer1* da figura 5.8, com o propósito de ser activado a cada 50 ms, indicando o início de um intervalo de canais (figura 3.4).

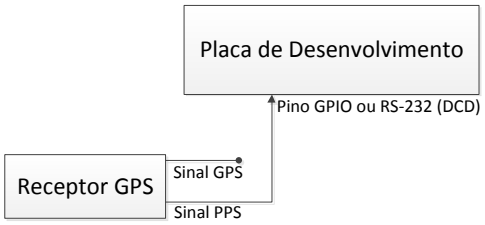


Figura 5.9: Esquema genérico da ligação do receptor GPS e a placa de desenvolvimento

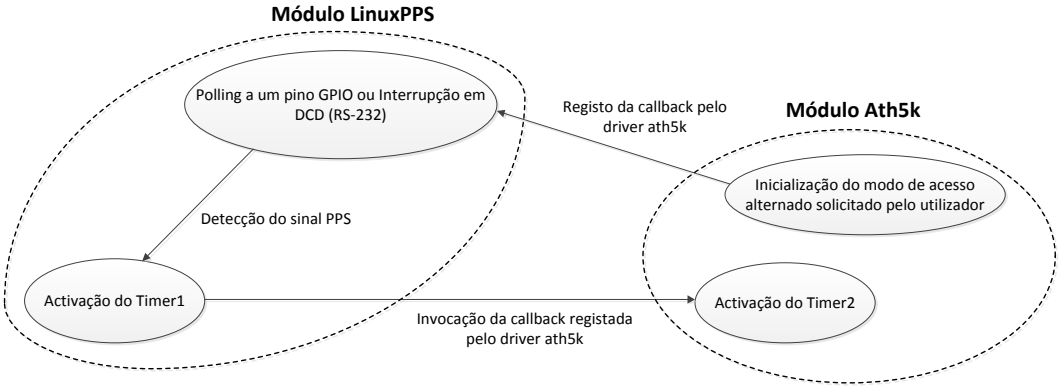


Figura 5.10: Interacção entre o *driver LinuxPPS* e o *driver ath5k*

O *output_timer* chama a função associada para o tratamento do mesmo, denominada *output_timer_event*. Esta função "rearma" o *output_timer* para 50 ms e chama a *callback* registada pelo *driver ath5k*. De modo a suportar esta *callback*, foi implementada uma API, denominada *output_timer_callback*, que guarda informação sobre o registo efectuado pelo *driver ath5k*, sendo que é inicializada a *NULL*. Para registar uma *callback* na API, foi implementada a função *output_timer_set_callback*. Esta função tem como entradas a *callback* que o *driver ath5k* pretende registar e a variável de entrada para a *callback* que pertence ao *driver ath5k*. Assim, teremos duas variáveis:

- **static void (*output_timer_callback)(void *data, int on_sec)**: Nesta API ficará registada a *callback* indicada pelo *driver ath5k*;
- **static void *output_timer_callback_data**: Nesta variável ficará guardada a variável de entrada para a *callback* respeitante ao *driver ath5k*.

Quando a função *output_timer_event* é chamada, a cada 50 ms, irá utilizar a *callback* registada da seguinte maneira:

output_timer_callback(output_timer_callback_data, on_sec)

Note que a variável de entrada *on_sec* é um valor inserido pelo *driver LinuxPPS* onde indica que este é o início de um segundo, se for diferente de 0. Isto é feito devido ao facto de o primeiro intervalo de canal num modo de acesso alternado ser o intervalo do CCH, ou seja, devemos garantir que a interface configurada no hardware no início de um segundo corresponde à interface configurada no CCH.

Como se pode perceber da descrição anterior, foi necessário implementar uma função para o registo da *callback* no *driver ath5k* quando o utilizador pretende inicializar uma comunicação com modo de acesso alternado no SCH e para apagar o mesmo registo quando o utilizador pretende voltar a ter uma comunicação com modo de acesso fixo no CCH ou SCH. A função de registo foi denominada *ath5k_register_switch_gps* (ficheiro *base.c*) e consiste na chamada da função *output_timer_set_callback*, com os parâmetros de entrada *ath5k_reset_auto_switch_channel* (ficheiro *base.c*), função a ser chamada quando o *Timer1* é activado (figura 5.8), e um ponteiro para a estrutura *ath5k_softc*. Assim, sempre que o *Timer1* for activado, a função *ath5k_reset_auto_switch_channel* é chamada com o parâmetro de entrada o ponteiro para a estrutura *ath5k_softc*. A função para apagar o registo anterior foi denominada *ath5k_unregister_switch_gps* (ficheiro *base.c*) e consiste na chamada à função *output_timer_set_callback* com os parâmetros de entrada a *NULL*.

A função *ath5k_reset_auto_switch_channel* consiste primeiramente na verificação da variável *on_sec*, garantindo que a variável de entrada *data* refere-se à interface configurada no CCH. A função *ath5k_reset_auto_switch_channel* é definida da seguinte forma:

static void ath5k_reset_auto_switch_channel(void *data, int on_sec)

A variável *data* corresponde à um ponteiro para uma estrutura *ath5k_softc*, onde foi criado o campo *act*, como vimos anteriormente em 5.5, sendo possível a partir do mesmo controlar qual das interfaces encontram-se activas no momento. Esta função além de garantir que a interface activa no início do segundo correponde à interface configurada no CCH, activa o *Timer2* (figura 5.8), destinado ao controlo do intervalo de guarda, como vimos em 5.5, efectuando a troca entre a interface activa e a inactiva, trocando assim entre o SCH e o CCH, ou vice versa.

5.7 Primitivas e a aplicação *iw*

De modo a disponibilizar ao utilizador as ferramentas necessárias à configuração das funcionalidades implementadas a partir do *userspace*, foi alterada a aplicação *iw*. A norma IEEE 1609.4 especifica uma série de primitivas que serão utilizadas pelas camadas superiores de modo a comunicar à sub-camada MAC como o sistema deve estar configurado. Com base nestas primitivas, implementaram-se comandos no *iw* de modo a estas serem acedidas pelo utilizador e podermos configurar o sistema da maneira mais cómoda, permitindo a realização de testes que mostram as funcionalidades implementadas. Note que os nomes dos ficheiros apresentados entre parênteses, "()", neste capítulo, referem-se à lista de ficheiros da aplicação *iw* presente no anexo A.

5.7.1 A aplicação *iw* e a interacção *userspace-kernelspace*

A aplicação *iw* utiliza as API's *nl80211* implementadas pelo *driver* disponibilizado pela *Linux Wireless*. Esta aplicação permite a configuração do dispositivo *wireless* através das suas interfaces. Desta forma, a partir do *iw*, podemos adicionar/remover uma interface virtual *wireless*, inicializar uma interface num modo específico, configurar o canal/frequência, nível de potência de transmissão ou taxa de transmissão que necessitamos. Como foi referido anteriormente, o *iw* oferece a capacidade do utilizador utilizar apenas as API's *nl80211* em vez das API's WEXT utilizadas pela aplicação semelhante *iwconfig*.

iw utiliza uma *socket netlink* para a troca de mensagens com o *driver*. Para tal, aquando

da sua inicialização, o *iw* aloca uma *socket netlink* através da função disponibilizada no *kernel* *nl_handle_alloc* e conecta-se à *socket generic netlink*, através da função *genl_connect*. Depois de conectado, o *iw* aloca *cache generic netlink* e procura a família *nl80211*, registada pelo *driver*, conectando-se assim ao mesmo. Através desta abordagem, teremos acesso à família *netlink* criada pelo módulo *cfg80211*, tirando partido das funcionalidades do *generic netlink*, descritas anteriormente no capítulo 4, secção 4.2.3.

No *include* do *driver* utilizado, encontra-se no ficheiro *nl80211.h* uma série de comandos que são definidos consoante a necessidade de uma funcionalidade e associados cada um a uma *callback* implementada no *driver*. Estes comandos são utilizados para referenciar na mensagem *netlink* qual a *callback* que irá tratar a mesma aquando da sua recepção. O registo e a implementação destas *callbacks* encontram-se no ficheiro *nl80211.c* do módulo *cfg80211*.

As *callbacks* referidas acima são anexadas à família *netlink* através da estrutura *genl_ops*. Esta estrutura apresenta os seguintes campos relevantes:

- **cmd**: comando que deverá ser identificado na mensagem *netlink*;
- **policy**: política de validação de atributos. Os atributos são basicamente posições específicas na mensagem *netlink* onde inserimos informação que será posteriormente tratada pela função associada ao comando;
- **flags**: *flags* associadas ao comando. A única *flag* utilizada nesta implementação é a *flag* que confere permissão de administrador;
- **doit**: nome da *callback* que será associada ao comando.

Estes quatro campos são assim utilizados para anexar um novo comando à família *netlink*. A seguir apresenta-se um exemplo de uma entrada no *array nl80211_ops* de estruturas *genl_ops*, que associa o comando *NL80211_CMD_JOIN_IBSS* à função
nl80211_join_ibss:

.cmd = NL80211_CMD_JOIN_IBSS,
.doit = nl80211_join_ibss,
.policy = nl80211_policy,
.internal_flags = NL80211_FLAG_NEED_NETDEV_UP,

O campo *policy* refere-se ao *array* denominado *nl80211_policy*, definido no ficheiro *nl80211 .c*. Este corresponde a um *array* de estruturas do tipo *nla_policy* que apresenta os campos *type*, que corresponde ao tipo do atributo, e *len*, que corresponde ao tamanho do mesmo

se for necessário especificar. Cada índice do *array* corresponde a um atributo definido no ficheiro *nl80211.h*. A seguir apresenta-se um exemplo de uma entrada do *array nla_policy*, que identifica o atributo que confere espaço na mensagem para a introdução da frequência:

[NL80211_ATTR_WIPHY_FREQ] = { .type = NLA_U32 },

Esta entrada especifica que a frequência deverá ser um valor de 32 bits sem sinal. Com esta abordagem apenas necessitamos de indicar o identificador *NL80211_ATTR_WIPHY_FREQ* à função de inserção ou remoção de dados da mensagem que esta trata de aceder à posição previamente reservada para tal. Neste caso não foi necessário indicar o tamanho do atributo pelo campo *len*, visto que o próprio tipo já o indica.

O campo *internal_flags* indica que a utilização deste comando depende de algum factor indicado pela *flag* que inserimos no mesmo. No exemplo acima, a *flag* *NL80211_FLAG_NEED_NETDEV_UP* indica que este comando apenas poderá ser executado se a interface que será afectada pela execução do comando estiver activa. Para tal, antes da chamada da *callback* dada pelo campo *doit*, é verificado se a interface encontra-se activa.

A implementação de comandos foi precedida de um estudo e compreensão de como são criadas e enviadas as mensagens, tanto por parte da aplicação como por *kernelspace*. Para tal, foi alterado o comando *NL80211_CMD_GET_INTERFACE* para apresentar informações mais detalhadas do estado de uma interface. De seguida apresentam-se alguns exemplos de como inserir e retirar informação dos atributos da mensagem *netlink*.

A função *nl80211_get_interface* é chamada quando uma mensagem *netlink* é recebida com o comando *NL80211_CMD_GET_INTERFACE*. Esta função cria uma nova mensagem *netlink* para responder ao pedido de informação da interface, através da função *nlmsg_new*. De seguida, a função *nl80211_get_interface* chama a função *nl80211_send_iface* que preencherá a mensagem com a informação requerida. No final, a função *nl80211_get_interface* chama a função *genlmsg_reply* de forma a responder à entidade que enviou a mensagem anteriormente recebida.

A função *nl80211_send_iface* preenche a mensagem *netlink* através da chamada às funções de inserção *netlink*. A seguir apresentam-se três exemplos de diferentes atributos inseridos numa mensagem *netlink*:

NLA_PUT_STRING(msg, NL80211_ATTR_IFNAME, dev->name);

NLA_PUT_U32(msg, NL80211_ATTR_IFINDEX, dev->ifindex);

NLA_PUT(msg, NL80211_ATTR_MAC, ETH_ALEN, wdev->wiphy->perm_addr);

Como se pode observar, as três funções de inserção referem-se a tipos diferentes de da-

dos. *NLA_PUT_STRING* insere na posição dada por *NL80211_ATTR_IFNAME* da mensagem *msg*, o nome da interface dada pelo campo *name* da estrutura *net_device* que contém informação da interface virtual *wireless*. A função de inserção *NLA_PUT_U32* insere na mensagem o identificador da interface e a função de inserção *NLA_PUT* insere o endereço MAC do dispositivo *wireless*, sendo que esta função não especifica o tipo do atributo que está a ser inserido, sendo da responsabilidade do programador a especificação do tamanho do mesmo. No caso, como é um endereço MAC, o atributo tem o tamanho dado pela *flag ETH_ALEN* e esta informação é retirada do campo *perm_addr* da estrutura *wiphy* que contém informação sobre o hardware e a interface física, ilustrada na figura 5.3.

Do lado da aplicação *iw*, a informação acerca da interface é requerida através da utilização do comando *iw dev wlan0 info* ou apenas *iw dev*, corridos num terminal. A diferença entre os dois comandos é que o primeiro apresenta apenas informação acerca da interface de nome *wlan0* e o segundo apresenta informação acerca de todas as interfaces *wireless* registadas no sistema. Ao correr o primeiro comando num terminal, o iw irá identificar as palavras *dev*, *wlan0* e *info*, criando uma mensagem *netlink*. Ao identificar a palavra *dev*, este procurará o identificador da interface de nome *wlan0* através da função *if_nametoindex* e inserirá o mesmo na mensagem através da função de inserção *NLA_PUT_U32*. Ao identificar a palavra *info* este procurará na sua base de dados de comandos, um comando que tenha associado a palavra *info*. A função *handle_interface_info* (ficheiro *interface.c*) é responsável pelo tratamento deste comando e chama a função *nl_cb_set* para identificar a função *print_iface_handler* (ficheiro *interface.c*) como *callback* de retorno da mensagem. Ou seja, quando receber a mensagem de resposta vinda do *kernelspace* à mensagem que *iw* enviou a pedir informações, a função *print_iface_handler* será chamada e irá retirar toda a informação necessária da mensagem, imprimindo a mesma.

Para retirar a informação da mensagem, é necessário primeiramente retirar todos os atributos da mensagem, separando-os do cabeçalho da mensagem e outros campos irrelevantes. Para tal utiliza-se a função *nla_parse* que copia a zona destinada aos atributos para um *array* de estruturas do tipo *nlattr*. Assim, a partir dos identificadores para atributos definidos no ficheiro *nl80211.h*, como *NL80211_ATTR_WIPHY_FREQ*, podemos aceder a qualquer atributo que nos seja relevante e que esteja contido na mensagem. Para retirar da mensagem a informação no tipo que pretendemos utilizamos as funções *nla_get*.

int freq = nla_get_u32(tb_msg[NL80211_ATTR_WIPHY_FREQ]);

O exemplo acima mostra como retirar a frequência contida na mensagem na posição dada

por *NL80211_ATTR_WIPHY_FREQ*. Podemos então tratar os dados retirados da mensagem como quisermos, que neste caso consiste na impressão do valor da frequência no terminal que requisitou esta informação a partir do comando *NL80211_CMD_GET_INTERFACE*. Depois de analisados os procedimentos efectuados em *kernelspace* e a nível da aplicação, veremos como podemos utilizar os comandos existentes e os comandos inseridos por este trabalho através de exemplos da utilização dos mesmos. De seguida analisaremos como cada novo comando foi implementado e qual a sua relevância.

5.7.2 Exemplos de utilização da aplicação iw

A aplicação *iw* fornece vários comandos destinados à configuração e obtenção de informação do estado do sistema ou do hardware. Nesta secção veremos exemplos de como podemos utilizar alguns comandos básicos do *iw*, focando os comandos implementados para a interface entre o utilizador e a sub-camada MAC WAVE. Note que os exemplos aqui apresentados incluem as alterações realizadas no decorrer da implementação deste projecto.

A figura 5.11 apresenta informação sobre uma interface virtual wireless, no caso *wlan0*, configurada no CCH. Através deste comando podemos verificar o canal, potência de transmissão, frequência, tipo ou protocolo correspondente em que esta interface encontra-se configurada. Temos informação ainda acerca do endereço MAC do hardware a que esta interface está anexada e o índice com que foi registada no sistema.

```
DRIVEIN49-root@sbc:~$ iw wlan0 info
wlan0    Protocol: IEEE 802.11p/1609.4(WAVE) Type: wave
         Ifindex: 10 Frequency: 5890 Channel: 178 Tx Power: 23 dbm
         HwAddr: 00:0b:6b:22:6d:7b ESSID: off
DRIVEIN49-root@sbc:~$
```

Figura 5.11: Exemplo do comando *iw dev info*

Neste exemplo obtemos informação acerca de uma interface virtual wireless já registada com um certo tipo e nome, associada a uma interface física correspondente ao hardware de endereço MAC *00:0b:6b:22:6d:7b*. Ou seja, nesta fase temos o nosso sistema configurado, o que implica a criação da interface virtual wireless *wlan0* e configuração do canal e potência de transmissão. De modo a criar uma interface virtual wireless, primeiro temos que saber qual a interface física que corresponde ao hardware que queremos utilizar. Para tal, utilizamos o comando *iw list* para verificar as capacidades presentes em todas as interfaces físicas registadas no sistema. Através deste comando, conseguimos obter informação acerca das bandas de frequência que o hardware correspondente suporta, os tipos de interface virtual wireless que

podemos utilizar (como *ad-hoc* ou *wave*) ou até as taxas de transferência suportadas por cada banda.

Depois de verificarmos o nome da interface física registada para o hardware que desejamos utilizar, criamos a nossa interface virtual wireless com o nome que quisermos e tipo desejados. A figura 5.12 ilustra a criação de uma interface virtual wireless denominada *wlan1* que é anexada à interface física *phy1*. O tipo seleccionado corresponde ao modo *wave* implementado neste trabalho. Se quisermos apagar esta interface virtual wireless, basta correrm os comando *iw wlan1 del*.

```
DRIVEIN49-root@sbc:~$ iw phy1 interface add wlan1 type wave
DRIVEIN49-root@sbc:~$
```

Figura 5.12: Exemplo do comando *iw phy interface add dev_name type dev_type*

Depois de criada a interface virtual wireless, temos que configurar a mesma no canal que queremos. Antes de configurar o canal, devemos modificar o estado da nossa interface para *running*, tornando-a uma interface activa, correndo o comando *ifconfig wlan1 up*, onde devemos configurar o endereço IP se quisermos transmitir dados IP. Para configurar o canal podemos proceder de duas maneiras, configurando a partir da frequência ou do número do canal. A figura 5.13 ilustra estas duas hipóteses. Configurado o canal, podemos configurar a potência de transmissão correndo o terceiro comando indicado também na figura 5.13. Por este exemplo, teremos a nossa interface virtual wireless configurada no canal 180, frequência 5900 MHz, com uma potência de transmissão fixada nos 23 dBm. A *flag* indicada pode ser *limit*, em que o valor indicado corresponde ao limite máximo a que a potência de transmissão pode variar, *fixed*, sendo o valor indicado é o valor fixo que a potência de transmissão deve ser configurada, e *auto*, onde o valor indicado corresponde a uma sugestão do utilizador, assim cabe ao algoritmo de controlo a configuração da potência de transmissão que este entende ser a melhor. Note que o valor deve ser passado em mBm. Podemos verificar o estado de configuração da nossa interface através do comando *iw wlan1 info* ilustrado pela figura 5.14.

```
DRIVEIN49-root@sbc:/testes$ iw wlan1 set channel 180
DRIVEIN49-root@sbc:/testes$ iw wlan1 set freq 5900
DRIVEIN49-root@sbc:/testes$ iw wlan1 set txpower fixed 2300
DRIVEIN49-root@sbc:/testes$
```

Figura 5.13: Exemplo dos comandos *iw "dev" set channel "chan_num"*, *iw "dev" set freq "freq_MHz"* e *iw "dev" set txpower "flag" "txpower_mBm"*

```
DRIVEIN49-root@sbc:/testes$ iw wlan1 info
wlan1    Protocol: IEEE 802.11p/1609.4(WAVE) Type: wave
         Ifindex: 12 Frequency: 5900 Channel: 180 Tx Power: 23 dbm
         HwAddr: 00:0b:6b:22:6d:7b ESSID: off
DRIVEIN49-root@sbc:/testes$
```

Figura 5.14: Informação obtida através do comando *iw wlan1 info* após a configuração apresentada acima

Depois das configurações básicas da interface virtual wireless criada, veremos de seguida como utilizar as funcionalidades WAVE implementadas por este trabalho. Uma vez que como resultado da implementação deste trabalho temos duas interfaces físicas e consequentemente duas interfaces virtuais wireless, foi implementado o comando *iw wave active*. Todos os comandos referentes à implementação da sub-camada MAC WAVE são identificados pela palavra *wave* na construcção do comando, representando uma secção, como será discutido mais à frente na secção 5.7.4.

O comando *iw wave active* fornece-nos informação acerca da interface virtual wireless activa no momento. A figura 5.15 ilustra a informação obtida com a execução deste comando.

```
DRIVEIN49-root@sbc:/testes$ iw wave active
wlan0 (phy0):
Frequency (channel): 5890 MHz (178) BitRate: 3 Mbps TxPower: 23 dBm
Channel access mode: continuous on CCH
DRIVEIN49-root@sbc:/testes$
```

Figura 5.15: Informação obtida através do comando *iw wave active*

Através deste comando obtemos informação sobre qual a interface virtual wireless configurada neste momento, o canal em que se encontra, a potência de transmissão configurada e a taxa de transmissão, assim como o modo de acesso ao canal configurado no momento. No exemplo da figura 5.15 temos o nosso hardware configurado no CCH, com uma taxa de transmissão a 3 Mbps (Mega bits por segundo), potência de transmissão configurada a 23 dBm e em modo contínuo no CCH. Para configurar o nosso sistema num SCH é necessário primeiro registarmos o perfil do transmissor, visto que iremos utilizar dados IP para testes, uma vez que o protocolo WSMP ainda não foi implementado. Para tal, foi implementado o comando *iw wave registerprofile "channel" "txpower" "datarate" "adaptable"*. A figura 5.16 apresenta um exemplo do registo de um perfil para um SCH.

```
DRIVEIN49-root@sbc:/testes$ iw wave registerprofile 180 23 24
DRIVEIN49-root@sbc:/testes$
```

Figura 5.16: Exemplo de utilização do comando *iw wave registerprofile "channel" "txpower" "datarate" "adaptable"*

Neste exemplo, registamos o canal 180 com uma potência de transmissão de 23 dBm e taxa de transmissão a 24 Mbps. A *flag adaptable* indica se a taxa de transmissão e a potência de transmissão podem ser adaptadas ou são fixas. Neste exemplo, escolhemos não indicá-la, portanto, serão fixas. Para sabermos mais tarde os parâmetros que indicamos no perfil para um certo canal, foi implementado o comando *iw wave getprofile "sch_number"*. Através deste comando podemos consultar a entrada da tabela de perfis de transmissores correspondente a um certo canal. Neste exemplo, obtemos a informação ilustrada na figura 5.17.

```
DRIVEIN49-root@sbc:/testes$ iw wave getprofile 180
Transmitter Profile (chan: 180):
TX Power Level: 23 dBm (Not adaptable)
Data rate: 24 Mbps (Not adaptable)
DRIVEIN49-root@sbc:/testes$
```

Figura 5.17: Informação obtida através do comando *iw wave getprofile "sch_number"*

Para iniciarmos a comunicação num SCH, foi implementado o comando *iw wave schstart "sch_number" "chan_access"*. A figura 5.18 apresenta o exemplo da inicialização da comunicação no canal 180 em modo de acesso ao canal contínuo.

```
DRIVEIN49-root@sbc:/testes$ iw wave schstart 180 continuous
DRIVEIN49-root@sbc:/testes$
```

Figura 5.18: Exemplo de utilização do comando *iw wave schstart "sch_number" "chan_access"*

Para obtermos informação acerca da interface virtual configurada corremos o comando *iw wave active* para verificarmos a configuração feita anteriormente, como mostra a figura 5.19.

```
DRIVEIN49-root@sbc:/testes$ iw wave active
wlan1 (phy1):
Frequency (channel): 5900 MHz (180) BitRate: 24 Mbps TxPower: 23 dBm
Channel access mode: continuous on SCH
DRIVEIN49-root@sbc:/testes$
```

Figura 5.19: Informação obtida através do comando *iw wave active* após iniciar-se a comunicação no SCH

Podemos verificar pela figura 5.19, que todas as configurações foram efectuadas com

sucesso. Note que o modo de acesso ao canal apresenta-se agora como contínuo no SCH. De modo a terminarmos a comunicação num SCH, utilizamos o comando implementado neste trabalho, *iw wave schend "sch_number"*. A figura 5.20 ilustra a aplicação deste comando para o exemplo anterior.

```
DRIVEIN49-root@sbc:/testes$ iw wave schend 180
DRIVEIN49-root@sbc:/testes$
```

Figura 5.20: Exemplo de utilização do comando *iw wave schend "sch_number"*

Depois de executado este comando, a interface virtual wireless referente ao CCH volta a ser configurada como activa e o modo de acesso ao canal volta ao padrão contínuo no CCH, como podemos ver através do comando ***iw wave active***, onde a informação apresentada será idêntica à da figura 5.15.

5.7.3 Implementação das primitivas em kernelspace

Na secção 3.7 do capítulo 3 foram descritas as primitivas definidas na norma IEEE 1609.4. Estas primitivas foram implementadas nos ficheiros criados com o nome *mlmex_extension.c* e *mlmex_extension.h*. Estes ficheiros foram criados no módulo *cfg80211*, visto que as funções aqui descritas foram inseridas como *callbacks* para comandos que foram registados no *array nl80211_ops*, tornando os mesmos opções para comunicação via mensagens *netlink* para a família *nl80211*. As funções implementadas foram as seguintes:

- **mlmex_schstart_request**: função destinada à requisição de comunicação num SCH com um específico modo de acesso ao canal;
- **mlmex_schend_request**: função destinada à requisição para terminar a comunicação num SCH específico;
- **mlmex_registertxprofile_request**: função destinada à requisição para o registo de um perfil de transmissor para um SCH específico;
- **mlmex_deletetxprofile_request**: função destinada à requisição para eliminar um perfil de transmissor de um SCH específico;
- **mlmex_canceltx_request**: função destinada à requisição para cancelar a transmissão numa fila de transmissão específica.

Todas as funções implementadas contêm uma pesquisa inicial ao identificador da interface física correspondente à interface configurada num SCH para não correrrmos o risco de configurar a interface correspondente ao CCH por engano. Esta pesquisa é efectuada com recurso a um ciclo '*while*' em que pesquisamos os dispositivos registados através da função *cfg80211_rdev_by_wiphy_idx*. Esta função tem como entrada o identificador de uma interface física onde começamos por procurar o identificador "0", visto que por defeito os dispositivos são registados sempre a começar por "0". Comparamos então a frequência configurada em cada dispositivo registado e se esta for diferente de 5890 MHz (CCH) e estiver na banda de frequência inserida para 5.9 GHz, encontramos a interface correcta. Todas as primitivas implementadas verificam inicialmente se a interface encontra-se em modo *wave*, visto que apenas fazem sentido se estivermos a trabalhar neste modo.

5.7.3.1 mlmex_schstart_request

Para implementar a primitiva *mlmex_schstart_request* necessitamos que a mensagem *netlink* recebida contenha a frequência do SCH que será configurado e o modo de acesso ao canal. Para este último foi introduzido um novo atributo ao qual denominou-se *NL80211_ATTR_CHAN_ACCESS* do tipo *NLA_U32*. Antes de configurar o hardware no canal pretendido, é feita uma pesquisa na estrutura *transmitterProfileTable* de modo a verificar-se se o perfil para o canal já foi registado. Se este for encontrado, configuramos os diferentes parâmetros contidos na tabela. A configuração da frequência é efectuada com recurso à função *cfg80211_set_freq* (ficheiro *chan.c*) disponibilizada pelo *driver*. Através da função *ieee80211_set_tx_power* (ficheiro *cfg.c*) configuramos o nível de potência de transmissão. Esta última função tem que ser acedida através de uma das API's disponibilizadas pela estrutura *cfg80211_ops*, visto que estamos a chamar a função a partir do módulo *cfg80211* e está implementada no módulo *mac80211*. Esta estrutura é acedida pelo campo *ops* da estrutura *cfg80211_registered_device*.

A configuração do modo de acesso ao canal exigiu a implementação de novos campos na estrutura *ieee80211_vif* (ficheiro *mac80211.h*) e uma função para o tratamento do mesmo à qual chamou-se *mlmex_set_chan_access* (ficheiro *mlmex_extension.c*). A estrutura *ieee80211_vif* contém informação acerca da interface virtual, como o modo *wireless* e o endereço MAC, e foram adicionados os campos *curr_access* e *old_access*. De modo a melhor compreensão e leitura do código, foi criada uma nova enumeração para estes dois campos que descrevem o modo de acesso actual e anterior, respectivamente. *enum nl80211_chan_access* contém os

seguintes identificadores:

- **NL80211_CHAN_ACCESS_ALT**: identifica o modo de acesso ao canal alternado;
- **NL80211_CHAN_ACCESS_CONT_ON_CCH**: identifica o modo de acesso ao canal contínuo no CCH;
- **NL80211_CHAN_ACCESS_CONT_ON_SCH**: identifica o modo de acesso ao canal contínuo no SCH;
- **NL80211_CHAN_ACCESS_IMME**: identifica o modo de acesso ao canal imediato;
- **NL80211_CHAN_ACCESS_EXT**: identifica o modo de acesso ao canal extendido;

Para atingirmos os objectivos desta dissertação, foram implementados apenas os modos de acesso mais importantes para o funcionamento da sub-camada MAC, o acesso contínuo e o acesso alternado. Os parâmetros de entrada da função *mlmex_set_chan_access* são:

- ***struct net_device *dev***: Ponteiro para uma estrutura *net_device* que é parâmetro de entrada da função *IEEE80211_DEV_TO_SUB_IF*, a partir da qual obtemos uma estrutura *ieee80211_sub_if_data* referente à interface wireless destinada a operar num SCH;
- ***enum nl80211_chan_access access***: Modo de acesso ao canal a ser configurado;
- ***bool immediate***: Variável booleana que indica se o acesso será imediato ou não;
- ***unsigned int extended***: Variável inteira que indica quantos intervalos de tempo o intervalo do SCH será extendido.

As variáveis *immediate* e *extended* foram indicadas acima apenas para futuras alterações à implementação actual, visto que esta não suporta estes tipos de modo de acesso no momento. A função *mlmex_set_chan_access* basicamente compara o parâmetro de entrada *access* com o campo *curr_access*, acedido pelo campo *struct ieee80211_vif vif* através da estrutura *ieee80211_sub_if_data*. Assim, se o acesso actual for igual ao requisitado, a função retorna o erro *EALREADY* indicando que este modo já se encontra activo.

Depois de configurada a interface destinada ao SCH, faz-se uma pesquisa análoga à feita anteriormente, mas agora para a interface configurada no CCH. É chamada a função *mlmex_set_chan_access* para a alteração dos campos *curr_access* e *old_access* também para

a interface do CCH. Este procedimento facilita a obtenção de informação acerca do modo configurado actualmente, visto que se o dispositivo estiver configurado no modo alternado não é necessário fazer uma pesquisa pela interface configurada no SCH para obtermos informação acerca do modo de acesso ao canal que o dispositivo opera no momento.

De forma a controlar a partir do utilizador o modo de acesso ao canal, foi implementado no módulo *mac80211* uma função denominada *wave_change_switch_mode* (ficheiro *cfg.c*). Esta função foi definida como uma API *cfg80211_ops* (ver figura 4.2), de modo a poder ser chamada no módulo *cfg80211*. A função *wave_change_switch_mode* apresenta os seguintes parâmetros de entrada:

- ***struct net_device *dev***: Ponteiro para uma estrutura *net_device* que é parâmetro de entrada da função *IEEE80211_DEV_TO_SUB_IF*, a partir da qual obtemos uma estrutura *ieee80211_sub_if_data* referente à interface wireless destinada a operar num SCH;

- ***enum nl80211_chan_access access***: Modo de acesso ao canal a ser configurado;

- **int freq**: Frequência que desejamos configurar para o modo de acesso indicado por *access*. Se o modo de acesso for alternado, este valor indicará a frequência do SCH desejado. Se o modo de acesso for contínuo, *freq* indicará a frequência do CCH, indicando que o modo de acesso é contínuo no CCH, ou a frequência do SCH desejado, indicando que o modo de acesso é contínuo no SCH.

Vimos anteriormente em 5.6.1 que para iniciarmos a comutação de canais, ou de interfaces, temos que registar a nossa *callback* para ser chamada pelo módulo do *driver LinuxPPS*. De forma a indicarmos que queremos registar a *callback* para operarmos em modo de acesso alternado, foi necessário alterar uma das API's em *ieee80211_ops* de modo a uma função do módulo *ath5k* poder ser invocada no módulo *mac80211* e efectuar o registo da *callback*. A função que foi alterada denomina-se *ath5k_config* (ficheiro *mac80211-ops.c*). Esta função invoca diferentes funções de tratamento a alterações do sistema, indicadas pela *flag* dada como parâmetro de entrada denominada *changed*. Definimos, então, mais duas *flags* em *enum ieee80211_conf_changed*, às quais chamamos *IEEE80211_CONF_CHANGE_REG_SWITCH* e *IEEE80211_CONF_CHANGE_UNREG_SWITCH*. Na função *ath5k_config*, quando a *flag changed* detecta *IEEE80211_CONF_CHANGE_REG_SWITCH*, a função *ath5k_register_switch_gps* é chamada registando a *callback ath5k_reset_auto_switch_channel*, como vimos anteriormente

em 5.6.1. Quando *changed* detecta *IEEE80211_CONF_CHANGE_UNREG_SWITCH*, registamos *NULL* por vez da *callback* anterior, apagando o registo à mesma.

Para configurar o modo de acesso contínuo num canal específico, inserimos uma *flag* à qual chamamos *IEEE80211_CONF_CHANGE_SWITCH*. Quando o parâmetro de estrada *changed* corresponde a esta *flag*, o *Timer2* (figura 5.8) é activado apenas uma vez, ou seja, mudamos para a interface inactiva, trocando apenas uma vez de canal. Este procedimento é útil quando o dispositivo encontra-se em modo de acesso ao canal contínuo no CCH e o utilizador pretende ter um modo de acesso ao canal contínuo mas agora num SCH específico. Assim, passamos a *flag IEEE80211_CONF_CHANGE_SWITCH* em *changed* e apenas trocamos para a interface inactiva, satisfazendo o pedido do utilizador.

5.7.3.2 mlmex_schend_request

A primitiva *mlmex_schend_request* requer o número do SCH a ser terminado. Assim, no início da função verificamos se a frequência está presente na mensagem *netlink* e se esta corresponde à frequência configurada na interface do SCH. Verifica-se também se o dispositivo está a operar em modo *wave*. Depois das verificações iniciais e de encontrada a interface configurada no CCH, chama-se a função *wave_change_switch_mode* a partir do campo *change_switch_mode* do campo *const struct cfg80211_ops *ops* da estrutura *cfg80211_registered_device*, como vimos anteriormente. Assim, indica-se no parâmetro de entrada *freq* que queremos configurar o hardware com a interface cuja frequência corresponde a 5890 MHz.

Depois de comutada a interface, chamamos a função *mlmex_set_chan_access* para alterar os campos *curr_access* e *old_access* de modo a actualizar o modo de acesso para *NL80211_CHAN_ACCESS_CONT_ON_CCH*, tanto na interface configurada no CCH, como na interface configurada no SCH. O parâmetro de entrada *changed* da função *ath5k_config* corresponde a *IEEE80211_CONF_CHANGE_SWITCH*, trocando apenas uma vez de interface. Uma vez que o dispositivo estava a operar na interface correspondente ao SCH, este passa a estar configurado na interface correspondente ao CCH. Temos então o hardware configurado de modo contínuo no CCH.

5.7.3.3 mlmex_registertxprofile_request

A tabela de perfis de transmissor é preenchida na interface destinada a um SCH. Sendo assim, a função *mlmex_registertxprofile_request* começa por procurar a interface configurada num SCH e que seja do tipo *wave*. Depois desta pesquisa inicial, a mensagem *netlink* é

validada, visto que deve conter informação acerca da frequência, potência de transmissão e taxa de transmissão. Estes valores são retirados da mensagem, como vimos em 5.7.1.

Primeiro verifica-se se o canal passado na mensagem já possui uma entrada no *array transmitterProfileTable*, campo presente na estrutura *ieee80211_sub_if_data*. Se não, preenche-se a primeira entrada vazia, ou seja, que apresenta *dot4ServiceChannelNumber* igual a zero. Se já existir uma entrada com o canal passado na mensagem, então alteram-se os valores correspondentes para os que foram passados na mensagem *netlink*.

5.7.3.4 mlmex_deletetxprofile_request

Esta função funciona de maneira análoga à função *mlmex_registertxprofile_request* (5.7.3.3). Começa-se com uma pesquisa inicial à interface correspondente à interface destinada a um SCH e que esteja configurada em modo *wave*. A mensagem *netlink* recebida deve conter apenas a frequência correspondente ao canal a que o perfil de transmissor corresponde. Assim, é efectuada uma pesquisa no *array transmitterProfileTable* pelo canal passado na mensagem. Se encontrado, todos os campos da entrada correspondente ao mesmo são postos a zero. Se não for encontrada uma entrada para o canal, é retornado um erro.

5.7.3.5 mlmex_canceltx_request

Para a implementação da função *mlmex_canceltx_request* foi necessário a criação de um novo atributo para a mensagem *netlink*, visto que necessitamos de saber qual a categoria de acesso que o utilizador necessita de cancelar a transmissão. Foi criado o atributo *NL80211_ATTR_ACCESS_CATEGORY* do tipo *NLA_U32*. A mensagem *netlink* recebida deve conter, portanto, a frequência do canal SCH e a categoria de acesso, numa gama de 0 a 3.

Para cancelar a transmissão de dados numa fila de categoria de acesso específica, foi criada a função *ieee80211_cancel_tx* (ficheiro *cfg.c*). Esta função foi registada como campo *cancel_tx* da estrutura *cfg80211_ops* (ficheiro *cfg80211.h*). Assim, a partir do campo *ops* da estrutura *cfg80211_registered_device* podemos chamar esta função do módulo *mac80211*, no módulo *cfg80211*.

A função *ieee80211_cancel_tx* (ficheiro *cfg.c*) funciona como função intermédia entre módulos, visto que esta tem a tarefa apenas de chamar a função *ath5k_cancel_tx* do módulo *ath5k*. Sendo assim, a função *ieee80211_cancel_tx* faz a ponte entre o módulo *cfg80211*, onde se recebe a mensagem *netlink* com o pedido para cancelar a transmissão numa determinada categoria

de acesso, e o módulo *ath5k*, onde se efectua esta tarefa.

A função *ath5k_cancel_tx* (ficheiro *mac80211-ops.c*) foi registada como novo campo *cancel_tx*, da estrutura *ieee80211_ops* (ficheiro *mac80211.h*). A estrutura *ieee80211_ops*, como vimos anteriormente, permite a uma função do módulo *mac80211* a chamada a uma função do módulo *ath5k*, a qualquer instante, desde que esta esteja registada nesta estrutura.

A implementação da função *ath5k_cancel_tx* foi efectuada a partir da função já existente do *driver* chamada *ath5k_hw_release_tx_queue*. Esta função torna a fila de número passado como parâmetro de entrada inactiva. Assim, quando a função termina, a fila que indicamos passa para o estado inactivo através da *flag AR5K_TX_QUEUE_INACTIVE*. Claro está que esta função só pode ser chamada se a interface que queremos cancelar a transmissão esteja no modo de acesso contínuo num SCH ou alternado.

5.7.4 Comandos auxiliares

Em 5.7.3 abordamos as primitivas que foram implementadas de acordo com as normas WAVE. De modo a tornarmos a implementação mais prática, algumas funções auxiliares foram implementadas de forma a obtermos informação acerca do *driver*, assim como os comandos da aplicação *iw*, correspondentes a cada primitiva implementada. Claro está que estes comandos não existirão quando o protocolo WSMP e a WME estiverem implementados, visto que a inicialização de um SCH ou o registo de um perfil de transmissor, por exemplo, será feito de forma transparente pelas camadas superiores da sub-camada MAC, com recurso à WME. Assim, temos os seguintes comandos auxiliares:

- **iw wave schstart <channel> <access>**: Inicializa o SCH dado por *channel* com o modo de acesso correspondente a *access*;
- **iw wave schend <channel>**: Termina a comunicação no SCH dado por *channel*;
- **iw wave registerprofile <channel> <txpower> <datarate>**: Regista uma nova entrada na tabela de perfis de transmissor com os valores do número de canal *channel*, potência de transmissão *txpower* e taxa de transmissão *datarate*;
- **iw wave deleteprofile <channel>**: Apaga a entrada da tabela de perfis de transmissor correspondente ao SCH dado por *channel*;
- **iw wave canceltx <channel> <access category>**: Cancela a transmissão na fila de categoria de acesso *access category*, na interface configurada no SCH dado por *channel*;

- **iw wave getprofile <channel>**: Apresenta os valores da entrada da tabela de perfis de transmissor correspondente ao SCH dado por *channel*;

- **iw wave active**: Apresenta informação acerca da interface activa no momento.

Pode-se notar que os comandos *iw wave getprofile* e *iw wave active* foram implementados de forma a obtermos informação na consola acerca do sistema. Antes dos comandos propriamente ditos, foi implementada uma nova secção, a secção *wave*. Assim, a seguir à designação da aplicação, *iw*, é necessário indicar que queremos trabalhar em modo *wave*. Para tal, foi criado um novo ficheiro na pasta da aplicação *iw*, com o nome de *wave.c* (anexo A). De forma a que este ficheiro seja identificado como secção *wave*, foi inserido no início do mesmo a linha *SECTION(wave)*, utilizando a *macro SECTION* já existente. Assim, todo o comando *iw* seguido da palavra *wave* será automaticamente tratado por uma função presente no ficheiro *wave.c*.

Uma função é anexada a um comando a partir da *macro COMMAND*: **COMMAND(section, name, args, cmd, flags, idby, handler, help)**. A função *handle_register_profile* foi criada para tratar o comando *iw wave registerprofile*, sendo que a *macro* correspondente foi colocada da seguinte forma:

COMMAND(wave,registerprofile, "<channel> <txpower> <datarate>", NL80211_CMD_REGISTERTXPROFILE, 0, CIB_NONE, handle_register_profile, "Register a Transmitter Profile to send IP data. NOTE: Just WAVE mode interfaces are allowed")

Podemos verificar que temos a secção (*section*), *wave*, o nome do comando (*name*), *registerprofile*, os argumentos de entrada (*args*), "<channel> <txpower> <datarate>", o comando *netlink* propriamente dito (*cmd*), *NL80211_CMD_REGISTERTXPROFILE*, *flags* relacionadas com o comando, se existirem, o identificador relativo à interface (*idby*), a função de tratamento do comando (*handler*), *handle_register_profile*, e alguma frase que ajudará na utilização do comando se o utilizador correr o comando *iw help*. Note que ao indicar a *flag CIB_NONE*, estamos a indicar que este comando não necessitará de ter associado qualquer identificador referente a uma interface. É fácil perceber porque estes comandos foram implementados com esta *flag* associada, uma vez que a tarefa de encontrar a interface correcta é feita pelas *callbacks* em *kernelspace*, como descrito na secção 5.7.3.

O funcionamento de todos os comandos implementados é semelhante, visto que consiste no envio de uma mensagem *netlink*, como foi abordado em 5.7.1. Existe uma validação inicial,

quanto aos valores inseridos na consola, por exemplo, o canal deve estar inserido na gama 5.85-5.925 GHz.

No caso dos comandos *iw wave getprofile* e *iw wave active*, estes enviam pedidos ao *kernelspace* e configuram uma *callback* que irá tratar a mensagem de resposta *netlink* quando esta for recebida. Assim vamos ter a *macro COMMAND* configurada da seguinte maneira para o comando *iw wave active*:

COMMAND(wave,active, NULL, NL80211_CMD_ACTIVE, 0, CIB_NONE, handle_active_info, "Show information about the active interface NOTE: Just WAVE mode interfaces are allowed")

A função *handler_active_info* apenas configura a função *print_active* como *callback* da mensagem *netlink*, através da função *nl_cb_set*. Quando receber a resposta, a função *print_active* irá tratar os dados passados na mensagem *netlink*, apresentando-os da seguinte forma:

wlan0 (phy0):

Frequency (channel): 5890 MHz (178) BitRate: 3 Mbps TxPower: 23 dBm

Channel access mode: continuous on CCH

Através deste comando, obtemos informação bastante completa acerca da interface activa no momento. O comando *iw wave getprofile* foi implementado de forma análoga, em que a *callback* denomina-se *print_profile* e a informação apresentada na consola é, por exemplo, a seguinte:

Transmitter Profile (chan: 180):

Tx Power Level: 23 dBm (Not adaptable)

Data rate: 24 Mbps (Not adaptable)

Podemos observar que o comando corrido na consola foi o *iw wave getprofile 180*. Através deste comando podemos consultar os valores registados para o canal 180 anteriormente.

O comando *iw wave schstart "sch_number" "chan_access"* apresenta a necessidade de validação do modo de acesso ao canal inserido pelo utilizador, além da frequência dada por *sch_number*. Estas verificações são efectuadas pela função *handle_sch_start*, função chamada para o tratamento da mensagem para o comando respectivo. A *macro COMMAND* é configurada da seguinte maneira para o comando *iw wave schstart "sch_number" "chan_access"*:

COMMAND(wave, schstart, "<channel> <access> [<extended intervals>]", NL80211_CMD_SCHSTART, 0, CIB_NONE, handle_sch_start, "Start a service channel. Use 'access' as continuous, alternate, immediate or extended. In order to set immediate access include 'immediate' and extended access indicate the

number of channel intervals. NOTE: Just WAVE mode interfaces are allowed")

O comando *iw wave schend "sch_number"* chama a função *handle_sch_end*, que trata a mensagem para o comando respectivo. A *macro COMMAND* é configurada da seguinte maneira para o comando *iw wave schend "sch_number"*:

COMMAND(wave, schend, "<channel>", NL80211_CMD_SCHEND, 0, CIB_NONE, handle_sch_end, "End a service channel. NOTE: Just WAVE mode interfaces are allowed")

O comando *iw wave canceltx "channel" "access category"* chama a função *handle_cancel_tx*, que trata a mensagem para o comando respectivo. Esta função deverá fazer a validação do valor inserido pelo utilizador para o número do canal e a AC, dados por *channel* e *access category*, respectivamente, sendo que AC só pode ser aceite se for um número entre 0 e 3 inclusive. A *macro COMMAND* é configurada da seguinte maneira para o comando *iw wave canceltx "channel" "access category"*:

COMMAND(wave, canceltx, "<channel> <access category>", NL80211_CMD_CANCEL_TX, 0, CIB_NONE, handle_cancel_tx, "Cancel transmission on 'channel' to the given 'access category'. NOTE: Just WAVE mode interfaces are allowed")

O comando *iw wave deleteprofile "sch_channel"* chama a função *handle_delete_profile*, que trata a mensagem para o comando respectivo. Esta função deverá fazer a validação do valor inserido pelo utilizador para o número do canal. A *macro COMMAND* é configurada da seguinte maneira para o comando *iw wave deleteprofile "sch_channel"*:

COMMAND(wave, deleteprofile, "<sch_channel>", NL80211_CMD_DELETE TXPROFILE, 0, CIB_NONE, handle_delete_profile, "Delete a Transmitter Profile. NOTE: Just WAVE mode interfaces are allowed")

Com este conjunto de comandos, somos capazes de monitorar e configurar o nosso dispositivo *WAVE* da maneira que nos for mais favorável, sempre respeitando as normas IEEE 802.11p/1609.4.

Capítulo 6

Resultados

6.1 Introdução

Neste trabalho foram implementadas as funcionalidades descritas nas normas IEEE 802.11p /1609.4 em *Linux*. Para tal, foi utilizada a placa mini-PCI DCMA-86P2, que apresenta o *chipset* AR5414A, desenvolvido pela *Atheros*, como abordado em 2.3. De modo a suportar o sistema operativo *Linux* e a utilização da placa mini-PCI, foi utilizado o micro-PC desenvolvido pela *PC Engines*, denominado Alix3D3. A tabela 6.1 apresenta as especificações técnicas deste micro-PC.

Tabela 6.1: Especificações Técnicas do Alix3D3 [41]

Parâmetro	**Detalhe**
CPU	500 MHz AMD Geode LX800
DRAM	256 MB DDR DRAM
Armazenamento	*socket* CompactFlash
Expansão	2 mini-PCI, barramento LPC
Conectividade	1 porta Ethernet
I/O	porta série DB9, dual USB, VGA, fones de ouvido (*out*) / microfone (*in*)
Alimentação	conector DC ou POE passivo, min. 7V - máx. 20V

A implementação do trabalho descrito nesta dissertação, foi seguida por uma análise do comportamento do sistema perante as alterações efectuadas à sub-camada MAC. Assim,

foi efectuado um conjunto de testes de forma a serem obtidos resultados e avaliada esta implementação. Os resultados obtidos para o tempo médio de troca entre canais, taxas de transferência , perda de pacotes e atraso na transmissão e recepção de pacotes, foram fruto de testes realizados em laboratório, ou seja, em ambiente controlado. O teste relacionado com o alcance máximo obtido foi realizado em uma estrada, com linha de vista.

Primeiramente foram realizados testes relativos ao tempo que um dispositivo WAVE demora a realizar em média a troca entre as duas interfaces, isto é, a comutação de canais. De seguida, foram realizados testes relativos às taxas de transferência que conseguimos obter sem a necessidade de haver uma BSS associada. Verificou-se também a percentagem de perda de pacotes quando o dispositivo opera no modo de acesso alternado. De modo a verificar a influência do aumento de tráfego em um SCH no envio de mensagens de emergência, tanto no SCH como no CCH, foram efectuados testes com o objectivo de se obter o tempo que um pacote correspondente a uma mensagem de emergência demora entre a sua transmissão e a sua recepção, onde também foi verificado o efeito da QoS no mesmo. Por último, foi realizado o teste relativo à obtenção do alcance máximo obtido com esta implementação. Os testes aqui descritos foram efectuados com dois micro-PCs Alix3D3 equipados com uma placa DCMA-86P2 cada e com um cartão de memória *flash* de 8 GB. Foi instalado previamente o *Linux* de versão 2.6.32 e instalado o *driver ath5k*, em que o mesmo pode ser obtido através do *website* apresentado no anexo A. A instalação do *driver* consiste primeiro na sua compilação, que consiste na utilização do comando *sudo make install* num terminal, no directório onde o *driver* se encontra. Depois de compilado o *driver*, teremos que inicializar o módulo correspondente ao mesmo. Para tal, utiliza-se o comando *modprobe ath5k*, carregando assim o módulo *ath5k* no *kernel*. Para os testes que foram realizados em laboratório, as duas placas utilizadas se encontravam afastadas cerca de 1 metro uma da outra, como ilustra a figura 6.1.

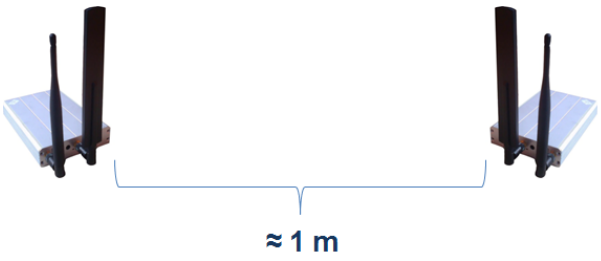


Figura 6.1: Disposição das placas para a realização dos testes em laboratório

6.2 Tempo Médio de Troca entre Canais

De modo a determinar o tempo médio de troca entre canais, ou entre interfaces no caso desta implementação, foi necessário implementar uma variável com o objectivo de guardar o tempo que o dispositivo demora a trocar entre interfaces. O tempo em *kernelspace* é medido por *ticks*. Um *tick* é o tempo entre duas expirações consecutivas do *timer* central do sistema. O *timer* central expira *HZ* vezes por segundo. O *HZ* é uma variável que depende do *kernel* utilizado, sendo que pode ser modificado, e, no caso deste trabalho, foi utilizado *HZ* igual a 1000. Isto significa que temos 1000 *ticks* a cada segundo, ou seja, o tempo entre duas expirações do *timer* central corresponde a 1 ms. A cada *tick* a variável global *jiffies* é incrementada, o que significa que o *jiffies* corresponde ao número de *ticks* ocorridos desde a inicialização do sistema [42]. Como queremos saber quanto tempo passou entre a ocorrência de um estado e outro do intervalo de guarda, o que temos que fazer é simplesmente guardar o valor do *jiffies* em uma variável local e comparar com o valor do *jiffies* posteriormente. Assim, quando um intervalo *MaxChSwitchTime* (ver figura 5.8), que corresponde ao estado *GI_MAX_SWITCH*, do intervalo de guarda é iniciado, guardamos o valor da variável *jiffies* naquele instante. Quando o estado *GI_SYNC_2_2* do intervalo de guarda é iniciado voltamos a guardar o valor da variável *jiffies*.

Para tal, foi implementada a estrutura *ath5k_switch_stat*, como ilustra a figura 6.2, e declarada a variável *ath5k_stat* como um *array* de estruturas *ath5k_switch_stat* de 20 posições, onde guardamos todos os tempos de troca entre interfaces de um segundo, visto que o dispositivo troca de canal a cada 50 ms. A estrutura *ath5k_switch_stat* apresenta os campos *switch_init* e *switch_end*, onde são guardados os valores de início de um intervalo *MaxChSwitchTime* e o final do mesmo, ou início do segundo intervalo de sincronização, como descrito acima. A partir destes dois campos conseguimos calcular o tempo de troca entre canais sendo a subtracção entre o campo *switch_end* e o campo *switch_init*. Temos que ter em atenção que *jiffies* é uma variável contadora que está sempre a ser incrementada, ou seja, tem um limite de 64 bits no caso do *kernel* utilizado para esta implementação. Dito isto, teremos que verificar se o valor armazenado em *switch_end* é maior do que *switch_init*. Os campos *switch_init* e *switch_end* foram definidos como *unsigned long* devido ao facto da variável *jiffies* ser definida como uma variável *unsigned* de 64 bits.

```
struct ath5k_switch_stat {
        unsigned long switch_init;
        unsigned long switch_end;
};
```

Figura 6.2: Estrutura *ath5k_switch_stat*

No início de um segundo, apresentamos a média dos valores guardados em *ath5k_stat*, a partir de um *printk*, em forma de *debug*, podendo ser visualizado através do comando *dmesg*, corrido num terminal. Os resultados obtidos foram muito semelhantes e depois de várias verificações, concluimos que em média um dispositivo WAVE, com as especificações físicas utilizadas nesta implementação, demora cerca de 3 ms a efectuar a troca entre canais, ou interfaces.

6.3 Taxas de Transferência

O *driver ath5k* apresenta *flags* que são utilizadas para configurar a taxa de transferência necessária no hardware, indicando à camada física o valor da *flag*. Estas *flags* estão definidas no ficheiro *ath5k.h*, possibilitando a configuração do hardwarer com os seguintes valores:

- ***ATH5K_RATE_CODE_6M***: Configura o hardware com uma taxa de transferência de 6 Mbps;
- ***ATH5K_RATE_CODE_9M***: Configura o hardware com uma taxa de transferência de 9 Mbps;
- ***ATH5K_RATE_CODE_12M***: Configura o hardware com uma taxa de transferência de 12 Mbps;
- ***ATH5K_RATE_CODE_18M***: Configura o hardware com uma taxa de transferência de 18 Mbps;
- ***ATH5K_RATE_CODE_24M***: Configura o hardware com uma taxa de transferência de 24 Mbps;
- ***ATH5K_RATE_CODE_36M***: Configura o hardware com uma taxa de transferência de 36 Mbps;

- ***ATH5K_RATE_CODE_48M***: Configura o hardware com uma taxa de transferência de 48 Mbps;

- ***ATH5K_RATE_CODE_54M***: Configura o hardware com uma taxa de transferência de 54 Mbps;

Como o hardware utilizado neste trabalho foi configurado inicialmente com 10 MHz para largura de banda (capítulo 5 secção 5.2), as taxas de transferência configuradas serão reduzidas para metade do seu valor. Sendo assim, teremos valores teóricos de 3, 4.5, 6, 9, 12, 18, 24, e 27 Mbps. De modo a controlarmos qual *flag* é registada para ser configurada no hardware, alteramos a função *ath5k_txbuf_setup* (ficheiro *base.c*), inserindo a taxa de transferência que desejamos na função *ath5k_hw_setup_mrr_tx_desc* (ficheiro *desc.c*), que indica ao descritor destinado ao controlo da transmissão a taxa de transferência que deve ser configurado.

De modo a medir qual a taxa de transferência real conseguida com cada *flag* configurada, foi utilizada a aplicação *iperf*. Todos os testes foram realizados por *User Datagram Protocol* (UDP), visto que queremos medir a taxa de envio, não sendo necessário resposta por parte do receptor. Assim, no receptor (servidor) foi utilizado o seguinte comando:

iperf -s -u

A opção *-s* indica que este corresponde ao servidor, visto que este auscultará o meio à espera de receber algo enviado pelo cliente. A opção *-u* indica que este apenas receberá pacotes enviados por UDP. No transmissor (cliente) foi utilizado o seguinte comando:

iperf -c <endereço de IP> -u -b 100M

Através deste comando enviamos dados por UDP, ocupando um *bitrate* de 100 Mbps, o que no nosso caso indica que queremos que envie à máxima taxa de transferência possível. Este valor foi escolhido por ser um valor que sabemos que o nosso hardware não consegue atingir, forçando o mesmo a enviar ao máximo que este consegue. A opção *-c* indica que este corresponde ao cliente, que envia os dados por UDP à máxima taxa de transferência conseguida para o endereço de IP indicado no comando. A opção *-b* corresponde ao *bitrate*, ou a taxa de transferência que queremos enviar os nossos dados.

Posto isto, os valores obtidos após configuração da *flag* de hardware correspondente são os apresentados na tabela 6.2.

Tabela 6.2: Taxas de transferência reais por *flag* de hardware, com a largura de banda configurada a 10 MHz

Flag	Taxa de Transferência (Mbps)	Taxa de Transferência Medida (Mbps)
ATH5K_RATE_CODE_6M	3	2.69
ATH5K_RATE_CODE_9M	4.5	3.87
ATH5K_RATE_CODE_12M	6	4.96
ATH5K_RATE_CODE_18M	9	6.95
ATH5K_RATE_CODE_24M	12	8.63
ATH5K_RATE_CODE_36M	18	11.5
ATH5K_RATE_CODE_48M	24	13.6
ATH5K_RATE_CODE_54M	27	14.8

6.4 Alcance Máximo

Um factor importante na análise da comunicação entre veículos consiste no alcance máximo que um dispositivo apresenta. Assim, foi realizado um teste onde foram utilizadas duas placas com as características apresentadas anteriormente, em que uma foi colocada estaticamente num ponto da estrada e a outra foi colocada a bordo de um veículo, como mostra a figura 6.3.

O teste consistiu no afastamento do veículo, enquanto foi efectuado um *ping* para a placa estática. Através da utilização de um receptor GPS, foram obtidas as coordenadas de cada placa, permitindo a obtenção da diferença entre as duas posições com precisão. Vericou-se então que se perdeu comunicação entre as duas placas a uma distância de aproximadamente 1115 metros. O teste foi repetido cinco vezes, sendo que foram obtidos valores muito próximos deste.

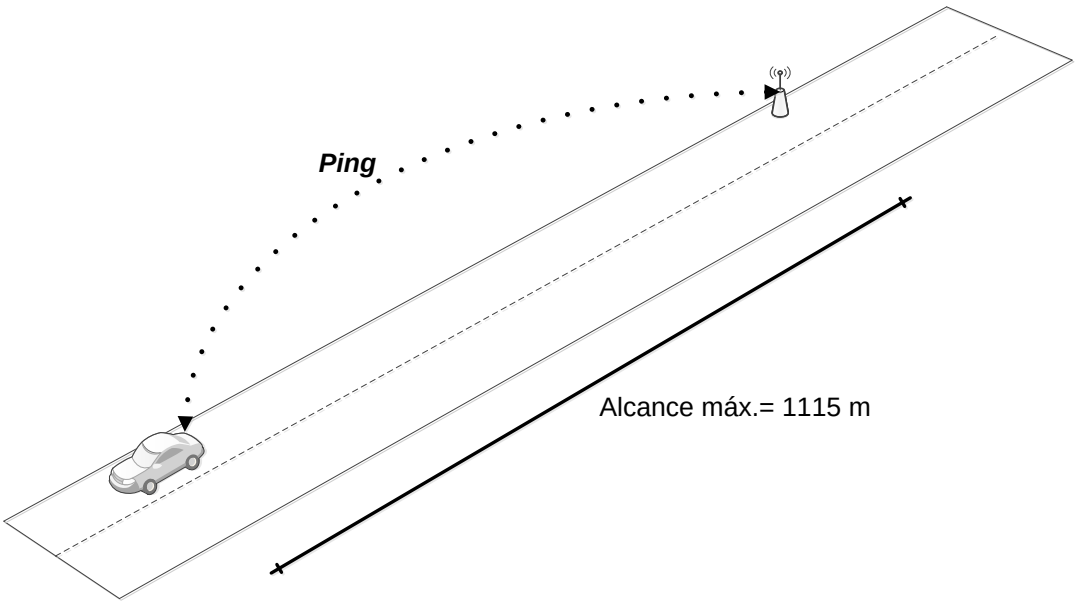


Figura 6.3: Ilustração do procedimento experimental destinado à obtenção do alcance máximo

6.5 Perda de Pacotes e Atraso na Transmissão e Recepção de Pacotes

Como foi referido anteriormente, o trabalho descrito nesta dissertação insere-se no projecto DRIVE-IN. O grupo de trabalho responsável pela implementação da sub-camada MAC implementou um filtro de Kalman que é utilizado para estimar o tempo UTC a partir do relógio interno do dispositivo e do pulso obtido pelo receptor GPS. Este filtro foi implementado de modo a seguir a sugestão apresentada no anexo F da norma IEEE 1609.4, sendo o filtro de Kalman considerado um estimador óptimo devido as características que este apresenta. Assim, foi incrementado o valor definido para a constante *HZ* no *kernel* do *Linux* para 2000, diminuindo assim o intervalo de tempo entre actualizações da variável *jiffies* para 0.5 ms (ver secção 6.2). Com esta abordagem diminuimos o intervalo de amostragem, diminuindo também o erro de amostragem [43].

A norma IEEE 1609.4 define que a qualidade temporal corresponde à estimativa do desvio padrão do erro do UTC estimado internamente. Assim, verificou-se que este erro corresponde a 70.8% do intervalo de tempo entre *ticks* (0.5 ms), que é igual a 0.35 ms. Com o filtro de Kalman verifica-se uma diminuição do mesmo para aproximadamente 0.15 ms.

Com esta implementação verificou-se que a perda de pacotes em modo de acesso alternado são de 0%, através do envio de 100000 pacotes utilizando tanto o *ping flood*, como *ping adaptative*.

De modo a avaliar a influência de um grande fluxo de tráfego diferente das mensagens de emergência, em paralelo com as mesmas, foi injectado tanto no SCH como no CCH, pacotes com o *ethertype* referente às WSM com a prioridade mais alta. Ao mesmo tempo, foi injectado no SCH um fluxo referente a um tráfego IP com uma prioridade diferente do mesmo, mais baixa.

O cenário utilizado é semelhante ao ilustrado pela figura 6.1, onde uma das placas permanece a transmitir nos dois canais, enquanto a outra apenas recebe os mesmos. Ambas as placas operam em acesso alternado.

Para simular as mensagens de emergência, WSM, foram criados dois ficheiros servidor e cliente, onde foram abertas *sockets raw*. Este tipo de sockets permitem escolher o destino através do endereço MAC, em vez do endereço IP, onde podemos também escolher qual o *ethertype* que o nosso pacote apresenta. Assim, inserimos o *ethertype* referente ao WSMP, simulando a transmissão de WSM's. Atribuimos ao pacote a prioridade mais alta. Transmitimos as WSM em *broadcast*, com uma taxa de transferência igual a 50 KB/s. Esta taxa de transferência foi escolhida de modo a evitar a perda de pacotes, visto que as filas de transferência podem encher rapidamente, descartando pacotes sem sequer transmití-los [43].

A injecção dos pacotes IP no SCH foi efectuada através da aplicação *iperf*, de forma semelhante à referida na secção 6.3. Analisamos assim a influência da ocupação do canal no atraso na recepção das WSM, transmitidas nos dois canais.

A figura 6.5 apresenta os valores do atraso medido através da captura dos pacotes referentes às WSM's, com a utilização da aplicação *tshark*, à medida que se aumenta a taxa de transferência a que são transmitidos os pacotes IP no SCH.

De modo a avaliar o mecanismo que confere QoS implementado no software utilizado neste trabalho, foi utilizada a aplicação *mgen* para gerar dois fluxos, um com a mais alta prioridade e outro com a mais baixa prioridade (*best-effort*). O fluxo com mais alta prioridade foi transmitido com taxa de transferência constante e o fluxo com mais baixa prioridade foi transmitido variando a sua taxa de transferência até o máximo de ocupação do canal. O cenário utilizado é semelhante ao ilustrado pela figura 6.1.

A figura 6.4 mostra a incluência no atraso na recepção de pacotes de alta e baixa prioridade, consoante a variação da taxa de transferência, ou seja, perante a ocupação do canal.

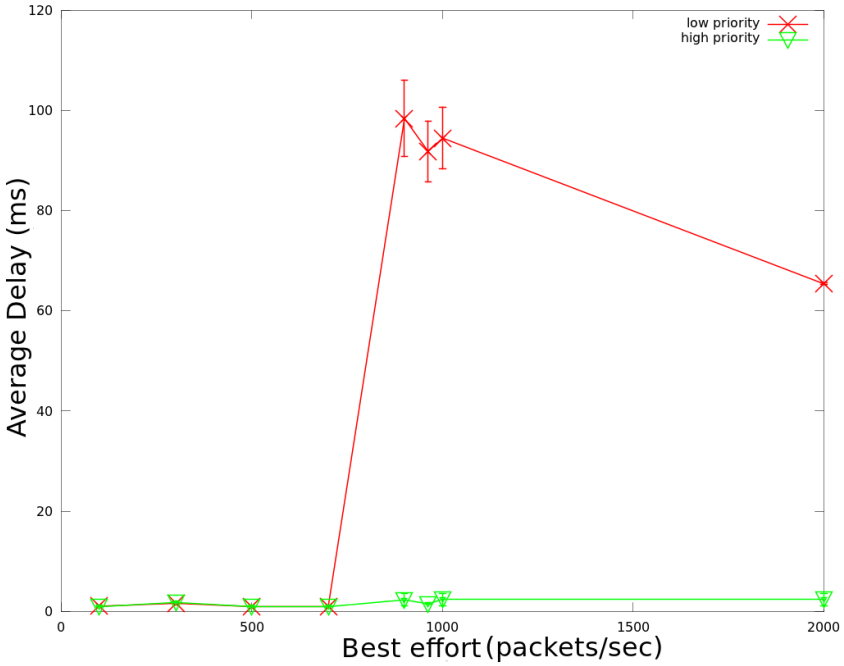


Figura 6.4: Comportamento de dois fluxos com diferentes prioridades [43]

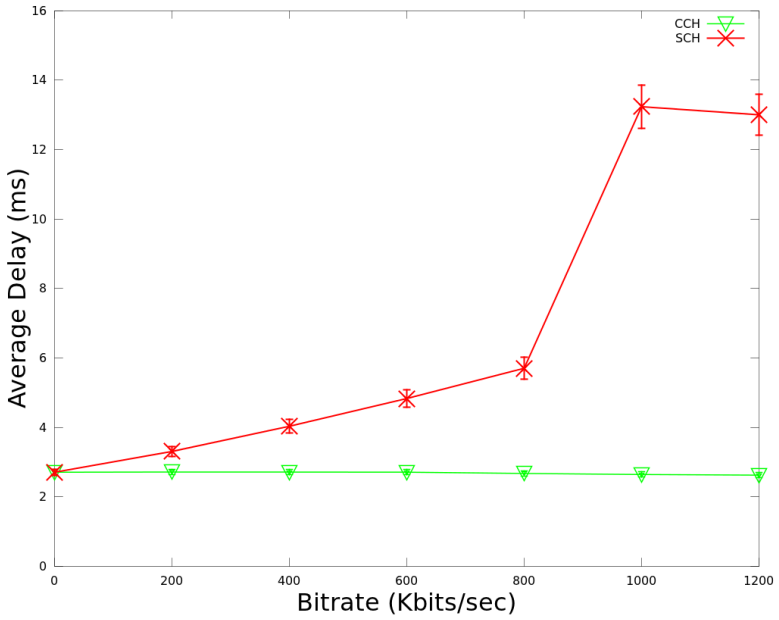


Figura 6.5: Influência da ocupação do SCH no atraso na recepção de WSM's [43]

Capítulo 7

Conclusão e Trabalho Futuro

7.1 Introdução

Neste capítulo apresentamos as conclusões que podemos tirar depois da análise da implementação, assim como os resultados dos testes efectuados, deste trabalho. Iremos discutir também o trabalho futuro, melhorias ao sistema e o seguimento de projectos que dependem da implementação apresentada por este trabalho.

7.2 Conclusão

O trabalho descrito nesta dissertação implementou todas as funcionalidades principais da sub-camada MAC WAVE, tirando partido da implementação da sub-camada MAC já existente, disponibilizada pelo grupo *Linux Wireless*. Depois de todo o estudo e compreensão das normas e do código disponíveis, foram implementadas as alterações à sub-camada MAC, descritas nas normas IEEE 802.11p/1609.4.

As conclusões que podemos discutir desta implementação, residem na análise dos resultados e em se o objectivo principal proposto por este trabalho foi alcançado, através da discussão do comportamento do sistema. O objectivo principal do trabalho descrito por esta dissertação, consiste na implementação da sub-camada MAC descrita pelas normas IEEE 802.11p/1609.4 em *Linux*, dispondo de um suporte de hardware previamente comprado. Depois de analisados os vários pontos que implementam a sub-camada MAC, podemos concluir que conseguimos alcançar o objectivo principal proposto, uma vez que todas as funcionalidades principais apresentadas em ambas as normas foram implementadas com sucesso, assim como o suporte para que o utilizador em *userspace*, seja capaz de configurar o sistema como

entender.

Assim, este trabalho oferece uma implementação que respeita no seu todo as normas IEEE 802.11/802.11p/1609.4. A implementação em *Linux* oferece ainda a capacidade desta implementação poder ser utilizada em qualquer placa que suporte a utilização do *Linux* e que apresente um barramento PCI para a utilização de uma placa mini-PCI que apresente o *chipset* da *Atheros* utilizado neste projecto.

7.2.1 Análise dos Resultados

Os resultados apresentados no capítulo 6, apresentam valores muito satisfatórios, tendo em conta o que podemos comparar na realidade e o que especificam as normas IEEE 802.11p/ 1609.4. Primeiro podemos concluir que o tempo médio de comutação entre dois canais é aceitável, visto que os 3 ms alcançados não ultrapassam os 4 ms que corresponde ao intervalo de guarda descrito na norma IEEE 1609.4, apesar de o tempo de troca de canais especificado nesta norma ser 2 ms. Podemos suprimir algum tempo dos dois intervalos destinados à análise da tolerância na sincronização, visto que podemos acelerar o processo, cabendo ao módulo do *LinuxPPS* a verificação se o dispositivo encontra-se sincronizado.

Em relação às taxas de transferência reais obtidas com este novo *driver* alterado, podemos concluir que as alterações implementadas não influenciaram negativamente sistema, uma vez que foram feitos testes anteriores às taxas de transferência para o modo *ad-hoc* normal com uma BSS criada, e os valores obtidos foram muito próximos.

O alcance máximo obtido é muito satisfatório, visto que para o DSRC está especificado que este deve ser capaz de comunicar no máximo à uma distância de aproximadamente um quilómetro [44].

Verificamos que a implementação da função de comutação de canais que confere o modo de acesso alternado ao dispositivo, apresenta uma perda de pacotes insignificante, visto que verificou-se uma perda de 0% na transmissão de 100000 pacotes. Este resultado avalia a conectividade entre os nodos em modo de acesso alternado que apresenta-se também muito satisfatória.

Com os testes relativos ao atraso na recepção de pacotes em relação à ocupação do SCH, verificou-se que o CCH é independente da ocupação do mesmo, visto que não apresenta perdas e o atraso entre os pacotes é igual independentemente da ocupação do SCH. No SCH verifica-se que as WSM simuladas apresentam um aumento no atraso na recepção das mesmas com a ocupação do canal. Verifica-se então que a independência entre os dois canais colabora

com a abilidade da transmissão de mensagens de emergência enquanto outros serviços são transmitidos no SCH.

Por fim, com os testes realizados para a verificação do mecanismo de QoS, verifica-se que o fluxo referente aos pacotes com mais alta prioridade não sofrem, ou quase não sofrem, influência pela presença do fluxo de mais baixa proridade. Verifica-se ainda que o atraso na recepção dos pacotes de mais baixa prioridade aumentam consideravelmente quando a ocupação do canal apresenta-se próxima do seu máximo, o que era esperado. Assim, verificamos também o bom funcionamento do mecanismo de QoS implementado.

7.3 Trabalho Futuro

O trabalho posterior à implementação deste projecto, consiste em aprimorar a mesma, ou seja, implementar todos os restantes pontos descritos nas normas IEEE 802.11p/1609.4, ou continuar o trabalho percorrendo todas as camadas superiores do modelo OSI, assim como as normas que as descrevem.

7.3.1 Trabalho Remanescente na sub-camada MAC

O trabalho descrito garante as funcionalidades mais importantes e o funcionamento da sub-camada MAC WAVE. Os aspectos referidos nas normas IEEE 802.11p/16094 que não foram abordados nesta dissertação são os seguintes:

- *Trama VSA*: Todo o tratamento para o envio e recepção de tramas VSA's deve ser implementado quando se coloca a hipótese de implementação do protocolo WSMP e, consequentemente, da WME. Isto devido ao facto de todas as tramas *WAVE Service Advertisement* (WSA) serem enviadas como parte das tramas VSA's;
- *Trama TA*: Todo o tratamento para o envio e recepção de tramas TA's deve ser implementado depois de haver um estimador que possa processar a informação disponibilizada na trama TA;

7.3.2 Camadas Superiores do Modelo OSI

O trabalho implementado oferece o suporte necessário à implementação ou alterações de camadas superiores à sub-camada MAC no modelo OSI para comunicações veiculares. Assim, indicamos como trabalho futuro a camada superior, camada de rede, que consistem nos seguintes pontos:

- *WME*: O envio de tramas WSA e o tratamento da mesma é responsabilidade da WME. Também a MIB WAVE é mantida pela WME;
- *WSMP*: Protocolo destinado às comunicações veiculares. Este protocolo deve interagir com a sub-camada LLC e com a WME;

O desenvolvimento da entidade de gestão WME e do protocolo WSMP em *Linux*, deve comunicar com a sub-camada MAC implementada neste trabalho. Como o desenvolvimento destes dois blocos da arquitectura WAVE será efectuado em *Linux*, este deve seguir um método análogo ao utilizado na implementação dos protocolos já existente, ou seja, com recurso à utilização de *sockets*. Assim, teremos para cada bloco um módulo comunicante através de *sockets* entre si e com o *userspace*. O módulo referente à WME pode ter a necessidade de comunicar directamente com a MLME, através da chamada das primitivas descritas nesta dissertação, funcionando como funções de acesso externo ao módulo *mac80211*, que contém a implementação da MLME.

Anexo A

Lista de ficheiros

O download do *driver ath5k* pode ser efectuado a partir de [45]. Neste sítio encontramos as várias versões publicadas pelo grupo *Linux Wireless* até a data. A versão utilizada neste projecto corresponde a de 31 de março de 2011, última versão lançada no início da alteração efectiva do *driver*. Apesar de se ter utilizado sempre a mesma versão, as versões mais recentes foram sendo verificadas de maneira a serem aproveitadas alterações importantes à parte que diz respeito a esta dissertação. A árvore seguinte apresenta os directórios e ficheiros relevantes utilizados/modificados/criados na implementação desta dissertação.

```
compat-wireless-2011-03-31
├── drivers
│   └── net
│       └── wireless
│           └── ath
│               └── ath5k
│                   ├── ath5k.h
│                   ├── attach.c
│                   ├── base.c
│                   ├── base.h
│                   ├── desc.c
│                   ├── desc.h
│                   ├── dma.c
│                   ├── mac80211-ops.c
│                   ├── pci.c
│                   ├── phy.c
│                   ├── reg.h
│                   └── reset.c
└── include
    ├── linux
    │   ├── ieee80211.h
    │   └── nl80211.h
    └── net
```

```
        ├── cfg80211.h
        └── mac80211.h
    ├── net
    │   ├── mac80211
    │   │   ├── cfg.c
    │   │   ├── cfg.h
    │   │   ├── chan.c
    │   │   ├── driver-ops.h
    │   │   ├── ibss.c
    │   │   ├── ieee80211_i.h
    │   │   ├── iface.c
    │   │   ├── main.c
    │   │   ├── mlme.c
    │   │   ├── rate.c
    │   │   ├── rate.h
    │   │   ├── rc80211_minstrel.c
    │   │   ├── rc80211_minstrel.h
    │   │   ├── rc80211_minstrel_ht.c
    │   │   ├── rc80211_minstrel_ht.h
    │   │   ├── rc80211_pid.h
    │   │   ├── rc80211_pid_algo.c
    │   │   ├── rx.c
    │   │   ├── scan.c
    │   │   ├── sta_info.c
    │   │   ├── sta_info.h
    │   │   ├── tx.c
    │   │   ├── util.c
    │   │   ├── wme.c
    │   │   └── wme.h
    │   └── wireless
    │       ├── chan.c
    │       ├── core.c
    │       ├── core.h
    │       ├── ibss.c
    │       ├── mlme.c
    │       ├── mlme_extension.c
    │       ├── mlme_extension.h
    │       ├── nl80211.c
    │       ├── nl80211.h
    │       ├── reg.c
    │       ├── reg.h
    │       ├── regb.h
    │       ├── scan.c
    │       ├── sme.c
    │       └── util.c
```

A aplicação *iw* pode ser obtida a partir de [46]. A versão 0.9.22 era a mais recente na altura em que a implementação foi inicializada. Apresenta-se a lista de ficheiros desta versão de seguida,

sendo que o ficheiro *wave.c* foi criado por este trabalho:

```
iw-0.9.22
├── bitrate.c
├── connect.c
├── cqm.c
├── event.c
├── genl.c
├── ibss.c
├── info.c
├── interface.c
├── iw.c
├── iw.h
├── link.c
├── mesh.c
├── mpath.c
├── nl80211.h
├── offch.c
├── phy.c
├── ps.c
├── reason.c
├── reg.c
├── scan.c
├── sections.c
├── station.c
├── status.c
├── survey.c
├── util.c
├── version.c
└── wave.c
```

Bibliografia

[1] F.J. Martinez, J.-C. Cano, C.T. Calafate, and P. Manzoni. CityMob: a mobility model pattern generator for VANETs. 2008.

[2] F.J. Martinez, J.-C. Cano, C.T. Calafate, and P. Manzoni. A performance evaluation of warning message dissemination in 802.11p based VANETs. pages 221 –224, oct. 2009.

[3] IEEE Std 802.11p-2010 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, IEEE Std 802.11n-2009, and IEEE Std 802.11w-2009). pages 1 –51, 15 2010.

[4] IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999). pages C1 –1184, 12 2007.

[5] IEEE Standard for Wireless Access in Vehicular Environments (WAVE)–Multi-channel Operation. *IEEE Std 1609.4-2010 (Revision of IEEE Std 1609.4-2006)*, pages 1 –89, 7 2011.

[6] L. Morgan. Notes on DSRC & WAVE Standards Suite: Its Architecture, Design, and Characteristics. *Communications Surveys Tutorials, IEEE*, PP(99):1 –15, 2010.

[7] Aaron Balchunas. In *OSI Reference Model*, pages 1 –8, 2007.

[8] IEEE. SP3 - SINTECH. `http://www.safespot-eu.org/sp3.html`.

[9] NEC. NEC LinkBird-MX Version 3 - Test Platform for Evaluation of Vehicular Communications Protocols. 2008.

[10] NEC Car to Car Communication System Takes Pole Position with Industry Leaders. `http://www.nec.co.jp/press/en/0811/1301.html`, November 2008.

[11] Arada Systems. Multi-Role WAVE Platform for Dedicated Short Range Communication (DSRC).

[12] Huaqun Guo and Shen Tat Goh and Foo, N.C.S. and Qian Zhang and Wai-Choong Wong. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International, title=Performance evaluation of 802.11p device for secure vehicular communication*, pages 1170 –1175, july 2011.

[13] Arada Systems. LocoMate RSU with miniPCI. `http://www.aradasystems.com/product-detail/locomate-rsu-with-minipci/`, 2010.

[14] Arada Systems. LocoMate OBU with miniPCI. http://www.aradasystems.com/product-detail/locomate-obu-with-minipci/, 2010.

[15] Kapsch. eWAVE - Embedded WAVE Module. http://www.kapsch.net/us/en/ktc/downloads/files/Kapsch-KTC-DS-IDS-e-Wave-05-EN-US.pdf.

[16] Kapsch. ITS Solutions - Portfolio. http://www.kapsch.net/us/en/ktc/portfolio/Pages/default_new.aspx.

[17] PRWeb. Kapsch Launches 5.9 GHz DSRC Module for On-Board Communications Systems. http://www.prweb.com/releases/itsworldcongress/wireless_communications/prweb1615614.htm.

[18] PRWeb. Kapsch Launches Next Generation MNCU R1551 for Transportation Infrastructure at 15th World Congress on Intelligent Transport Systems. http://www.prweb.com/releases/itsworldcongress/wireless_communications/prweb1615594.htm.

[19] Kapsch. Smarter Vehicles, Safer Roads MCNU R1500. http://www.kapsch.net/us/en/ktc/downloads/files/Kapsch-KTC-DS-MCNU-R1551-EN-US.pdf.

[20] CohdaWireless. Products - MK2 WAVE-DSRC Radio. http://www.cohdawireless.com/product/mk2.html.

[21] Robert Lasowski, Tim Leinmüller, Markus Strassberger, and Cirquent Gmbh. Openwave engine / wsu- a platform for c2c-cc.

[22] Unex - DCMA-86P2: 5.9GHz DSRC wireless mini-PCI, AR5414A-B2B. http://www.unex.com.tw/product/dcma-86p2, 2010.

[23] SAFESPOT. SAFESPOT Integrated Project. http://www.safespot-eu.org/.

[24] SAFESPOT. IEEE 1609 - Family of Standards for Wireless Access in Vehicular Environments (WAVE). http://www.standards.its.dot.gov/fact_sheet.asp?f=80.

[25] Jakob Eriksson, Hari Balakrishnan, and Samuel Madden. Cabernet: vehicular content delivery using wifi. In *Proceedings of the 14th ACM international conference on Mobile computing and networking*, MobiCom '08, pages 199–210, New York, NY, USA, 2008. ACM.

[26] Hong-Yu Huang, Pei-En Luo, Minglu Li, Da Li, Xu Li, Wei Shu, and Min-You Wu. Performance Evaluation of SUVnet With Real-Time Traffic Data. *Vehicular Technology, IEEE Transactions on*, 56(6):3381 –3396, nov. 2007.

[27] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. CarTel: a distributed mobile sensor computing system. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 125–138, New York, NY, USA, 2006. ACM.

[28] Carolina Pinart, Pilar Sanz, Iván Lequerica, Daniel García, Isaac Barona, and Diego Sánchez-Aparisi. DRIVE: a reconfigurable testbed for advanced vehicular services and communications. In *Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*, TridentCom '08, pages 16:1–16:8, ICST, Brussels, Belgium, Belgium, 2008.

[29] IT - Instituto de Telecomunicações - DRIVE-IN. http://www.it.up.pt/research/projects/current_prjects/current-projects.

[30] Saurabh Sehrawat and Revoti Prasad Bora and Dheeraj Harihar. Performance Analysis of QoS supported by Enhanced Distributed Channel Access (EDCA) mechanism in IEEE 802.11e.

[31] ath5k. http://linuxwireless.org/en/users/Drivers/ath5k.

[32] Existing Linux Wireless drivers. http://linuxwireless.org/en/users/Drivers.

[33] M. Vipin and S. Srikanth. Analysis of open source drivers for IEEE 802.11 WLANs. In *Wireless Communication and Sensor Computing, 2010. ICWCSC 2010. International Conference on*, pages 1 –5, jan. 2010.

[34] cfg80211. http://linuxwireless.org/en/developers/Documentation/cfg80211.

[35] netlink(7) - Linux man page. http://linux.die.net/man/7/netlink.

[36] Kevin Kaichuan He. Kernel Korner - Why and How to Use Netlink Socket. http://www.linuxjournal.com/article/7356, January 2005.

[37] Ariane Keller. Kernel Space - User Space Interfaces. http://people.ee.ethz.ch/~arkeller/linux/kernel_user_space_howto.html, July 2008.

[38] The Linux Foundation. Generic Netlink Howto. http://www.linuxfoundation.org/collaborate/workgroups/networking/genericnetlinkhowto, November 2009.

[39] Linux Wireless. iw. http://linuxwireless.org/en/users/Documentation/iw.

[40] Using a Garmin GPS 18 LVC as NTP stratum-0 on Linux 2.6. http://webcache.googleusercontent.com/search?q=cache:LVjb9Sj0ZvEJ:time.qnan.org/+%22time.qnan.org%22&cd=1&hl=pt-BR&ct=clnk&gl=pt.

[41] PC Engines - Alix3D3. http://pcengines.ch/alix3d3.htm.

[42] Benvenuti, Christian. *Understanding Linux Network Internals*. O'Reilly Media, Inc., 2005.

[43] Carlos Ameixieira, José Matos, Ricardo Moreira, Arnaldo Oliveira, André Cardote, and Susana Sargento. An IEEE 802.11p/WAVE Implementation with Synchronous Channel Switching for Seamless Dual-channel Access. *2011 IEEE Vehicular Networking Conference (VNC)*, pages 1–8, July 2011.

[44] Dr. Michele Weigle. *Standards: WAVE / DSRC / 802.11p*. 2008.

[45] Linux Wireless. Index of /download/compat-wireless-2.6/. http://linuxwireless.org/download/compat-wireless-2.6/.

[46] Linux Wireless. Index of /download/iw/. http://linuxwireless.org/download/iw/.